Acta Informatica manuscript No.
(will be inserted by the editor)

Naoki Kobayashi

Type-Based Information Flow Analysis for the Pi-Calculus

Received: date / Revised: date

Abstract We propose a new type system for information flow analysis for the π-calculus. As demonstrated by recent studies, information about whether each communication succeeds is important for precise information flow analysis for concurrent programs. By collecting such information using ideas of our previous type systems for deadlock/livelock-freedom, our type system can perform more precise analysis for certain communication/synchronization patterns (like synchronization using locks) than previous type systems. Our type system treats a wide range of communication/synchronization primitives in a uniform manner, which enabled development of a clear proof of type soundness and a sound and complete type inference algorithm.

Keywords type system – information flow analysis – pi-calculus – secrecy

1 Introduction

Information flow analysis is a static program analysis to check that a program does not leak information about secret data. Since Denning and Denning's work [5], information flow analysis has been studied for various programming languages, including imperative languages [5, 33], functional languages [7, 28], low-level languages [19, 35], and concurrent languages [10, 11, 27, 29, 31, 36].

Previous information flow analyses (especially, automated ones) for concurrent languages have not been quite satisfactory. Some of them [31] have shared memory primitives as only the communication/synchronization primitives. Although synchronization primitives can be expressed in terms of shared memory

Send offprint requests to: Naoki Kobayashi

N. Kobayashi
Graduate School of Information Sciences, Tohoku University, 6-3-9 Aoba, Aramaki, Aoba-ku, Sendai-shi, Miyagi 980-8579, Japan
E-mail: koba@ecei.tohoku.ac.jp

primitives by using, for example, Peterson's algorithm, the resulting type-based information flow analysis for such encoding is not precise enough. Honda and Yoshida [10, 11], Pottier [27], Hennessy and Riely [8, 9], and Zdancewic and Myers [36] studied information flow analysis for the π-calculus and similar calculi, so that various communication/synchronization primitives can be dealt with in a uniform manner. Pottier's type system [27] and Hennessy and Riely [8, 9] are, however, not expressive enough: Let us consider a process $x(\,).\,P$, which waits to receive a null tuple on channel x and then behaves like P. In Pottier's type system, if x is a secret (high-level) channel (a channel such that the communication behavior on it is kept secret; the precise meaning becomes clearer later in the paper), then P can communicate via only secret channels (since if P performs communication on non-secret channels, it may reveal information that someone has sent a message on the secret channel x). So, once a process performs communication or synchronization on a secret channel, it can no longer perform communication on non-secret channels, which is too restrictive. Honda and Yoshida [11] and Zdancewic and Myers [36] overcome this problem by allowing P to communicate via non-secret channels if the input on x always succeeds. For that purpose, they introduced special kinds of communication channels, and constructed type systems guaranteeing that communications on such channels always succeed, so that $x(\,).\,P$ is allowed as long as x is such a channel, even if x is secret and P communicates through non-secret channels. A problem of those type systems [11, 36] is, however, that they can deal with only specific kinds of channels (such as linear channels). For example, they cannot properly deal with processes using locks. Another problem of Honda and Yoshida's type system [11] is that there seems to be no reasonable type inference algorithm (which works as an algorithm for information flow analysis) for their type system: Since they introduce a separate typing rule for each kind of channel, a programmer at least needs to explicitly declare the kind of each channel to enable type inference.

In this paper, we propose a type system for information flow analysis for the π-calculus, which relaxes the above-mentioned limitations of information flow analyses for concurrent languages. As indicated above, to enable precise analysis of information flow for concurrent languages, it is important to analyze whether each communication/synchronization succeeds or not. While the previous work [11, 36] introduces specific kinds of channels to enable that analysis, our type system can treat general usage patterns of communication channels, using the ideas of our previous type system for lock-freedom [14]. One of the key ideas is to extend channel types with *channel usages*, which express how each channel is used for input/output and whether each input/output is guaranteed to succeed. Our type system can deal with various communication/synchronization patterns in a uniform manner, such as the cobegin/coend-style synchronization, locks (binary semaphores), and communications through linear channels [17]. Such uniform treatment of communication patterns also leads to a clear proof of the soundness of the type system. More detailed technical comparisons with previous work is found in Section 7.

The second main contribution of the present paper is the development of a sound and complete type inference algorithm (which works as an automatic algorithm for information flow analysis) for our type system for information flow analysis. As mentioned above, the type system in the present paper borrows ideas

from our previous type system for lock-freedom [14]. There was, however, no type inference algorithm for the latter type system. Following our previous work on type inference for a type system for deadlock-freedom [18] (which guarantees a weaker property that certain communications eventually succeed *unless the whole process diverges*), we extend channel usages of the previous type system [14] with recursion and choice operators. The previous type inference algorithm for deadlock-freedom [14] was sound but incomplete. Since the type system in the present paper and its (sound and complete) type inference algorithm can also be easily modified to obtain type systems and inference algorithms for deadlock-freedom and lock-freedom, the present work can also be considered a refinement of the previous work on deadlock-freedom and lock-freedom [14, 18]. Indeed, we have already implemented a tool that can automatically analyze lock-freedom, deadlock-freedom, and information flow for π-calculus processes, based on the algorithm described in this paper [16].

The rest of this article is structured as follows. Section 2 introduces the target language of our type-based information flow analysis. Section 3 presents our type system for information flow analysis, and Section 4 proves its soundness. Section 5 describes a type inference algorithm. In this article, we make several restrictions that are not present in the previous type systems for information flow analysis, so that our type system does not completely subsume the previous type systems (Honda and Yoshida's type system [11], in particular). Some of the restrictions are imposed just to clarify the essence (e.g., absence of subtyping on values) while others seem essential for the soundness of our type system. We discuss them in Section 6. Section 7 discusses related work, and Section 8 concludes.

2 Target Language

This section introduces the target language of our type-based information flow analysis. The language is a subset of the polyadic π-calculus [23], extended with booleans values and conditionals.[1]

2.1 Syntax

We first introduce secrecy levels, which denote the degree of secrecy of information about data and processes. For the sake of simplicity, we only consider two secrecy levels: **H**, which describes secret information (i.e., information that should be kept to privileged principals), and **L**, which describes non-secret information (i.e., information that can be revealed to any principal). As usual, if we need to deal with more than two secrecy levels, we can classify secrecy levels into **H** and **L** in many ways, and run the information flow analysis on the two levels for each classification.

Definition 1 (secrecy levels) The set of *secrecy levels* is $\{\mathbf{H}, \mathbf{L}\}$. The binary relation $\sqsubseteq$ on secrecy levels is the total order defined by $\mathbf{L} \sqsubseteq \mathbf{H}$.

[1] It is in principle possible to encode booleans and conditionals into the π-calculus, but a more complex type machinery (such as those studied in [13]) is necessary to perform a sufficient analysis for the encoding.

We use a meta-variable l for a secrecy level.

Definition 2 (processes) The set of processes, ranged over by P, is defined by:

$$\begin{array}{rl} P ::= & \mathbf{0} \mid \overline{x}\langle v_1, \ldots, v_n\rangle . P \mid x(y_1, \ldots, y_n) . P \\ & \mid (P \mid Q) \mid *P \mid (\nu x : \xi)\, P \mid \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q \\ v\ ::= & \mathit{true}^l \mid \mathit{false}^l \mid \star \mid x \end{array}$$

Here, x and y_i range over a countably infinite set $\mathbf{Var}$ of variables. ξ ranges over the set of core channel types (which is defined later in Section 3).

Notation 1 The prefix $x(y_1, \ldots, y_n)$ binds variables $y_1, \ldots, y_n$ and $(\nu x : \xi)$ binds x. As usual, we identify processes up to α-conversions (renaming of bound variables), and assume that α-conversions are implicitly applied so that bound variables are always different from each other and from free variables. We write $[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]P$ for the process obtained by replacing all the free occurrences of $x_1, \ldots, x_n$ in P with $v_1, \ldots, v_n$. We write $\tilde{x}$ for a sequence of variables $x_1, \ldots, x_n$. We abbreviate $[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]$ and $(\nu x_1 : \xi_1) \cdots (\nu x_n : \xi_n)$ to $[\widetilde{x} \mapsto \widetilde{v}]$ and $(\nu \widetilde{x} : \widetilde{\xi})$ respectively. We often omit $\mathbf{0}$ and write $\overline{x}\langle \widetilde{v} \rangle$ and $x(\widetilde{y})$ for $\overline{x}\langle \widetilde{v} \rangle . \mathbf{0}$ and $x(\widetilde{y}) . \mathbf{0}$ respectively.

We assume that prefixes ($\overline{x}\langle \widetilde{v} \rangle$, $x(\widetilde{y})$, (νx), and $*$) bind tighter than the parallel composition operator $\mid$, so that $\overline{x}\langle \widetilde{y} \rangle . P \mid Q$ means $(\overline{x}\langle \widetilde{y} \rangle . P) \mid Q$, not $\overline{x}\langle \widetilde{y} \rangle . (P \mid Q)$.

Process $\mathbf{0}$ does nothing. Process $\overline{x}\langle \widetilde{v} \rangle . P$ sends a tuple $\langle \widetilde{v} \rangle$ on x and then (after the tuple is received by some process) behaves like P. Each v_i is a boolean (true^l or false^l), the unit value ($\star$) (the element of a singleton set), or a variable. A communication channel is represented by a free variable or a variable bound by $(\nu x : \xi)$.

Process $x(\widetilde{y}) . P$ waits to receive a tuple $\langle \widetilde{v} \rangle$ on x and then behaves like $[\widetilde{y} \mapsto \widetilde{v}]P$. $P \mid Q$ represents concurrent execution of P and Q. $*P$ represents infinitely many copies of the process P running in parallel, and $(\nu x : \xi)\, P$ denotes a process that creates a fresh communication channel x and then behaves like P. The core channel type ξ is attached just for technical convenience (in proving soundness of the type system); It does not affect the operational semantics. We often omit ξ when it is not important. $\mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q$ behaves like P if v is true^l and behaves like Q if v is false^l; otherwise it is blocked forever.

The secrecy level l attached to a boolean value (true^l or false^l) expresses the degree of secrecy of the value. For example, information about $\mathit{true}^{\mathbf{H}}$ should not be revealed to non-privileged principals. On the other hand, no secrecy level is attached to $\star$, since it carries no information. The secrecy level of a communication channel may be specified in the core channel type ξ of $(\nu x : \xi)$. Those annotations of secrecy levels (for values and channels) are only used for a programmer to declare which value should be hidden to non-privileged principals. The programmer can omit them if they are unnecessary, since the type inference algorithm described in Section 5 can recover them.

In examples given in the rest of this paper, we sometimes annotate variables with their secrecy levels, just for the sake of readability.

2.2 Operational Semantics

Following the standard reduction semantics for the π-calculus, we define the operational semantics using a structural relation $P \preceq Q$, and a reduction relation $P \longrightarrow Q$. The former relation means that P can be restructured to Q by using the commutativity and associativity laws on $|$, etc. The latter relation means that P is reduced to Q by one communication on a channel. Differences from the standard reduction semantics are that $\preceq$ is not symmetric, and that we include reductions on if-expressions in $\preceq$ rather than in $\longrightarrow$ (so that $\mathbf{if}\ \mathit{true}^l\ \mathbf{then}\ P\ \mathbf{else}\ Q \preceq P$). The idea of using a non-symmetric structural relation goes back to our previous type system for deadlock-freedom [18]. That is necessary to make the type preservation hold in our type system.

Definition 3 The *structural preorder* $\preceq$ is the least reflexive and transitive relation closed under the following rules ($P \equiv Q$ denotes $(P \preceq Q) \wedge (Q \preceq P)$):

$$P \equiv P \mid \mathbf{0} \qquad \text{(S-ZERO1)}$$

$$\mathbf{0} \equiv *\mathbf{0} \qquad \text{(S-ZERO2)}$$

$$\mathbf{0} \equiv (\nu x : \xi)\, \mathbf{0} \qquad \text{(S-ZERO3)}$$

$$P \mid Q \equiv Q \mid P \qquad \text{(S-COMMUT)}$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad \text{(S-ASSOC)}$$

$$(\nu x : \xi)\, P \mid Q \equiv (\nu x : \xi)\, (P \mid Q) \quad \text{(if } x \text{ is not free in } Q\text{)} \qquad \text{(S-NEW)}$$

$$(\nu x : \xi_1)\, (\nu y : \xi_2)\, P \equiv (\nu y : \xi_2)\, (\nu x : \xi_1)\, P \qquad \text{(S-SWAP)}$$

$$\mathbf{if}\ \mathit{true}^l\ \mathbf{then}\ P\ \mathbf{else}\ Q \preceq P \qquad \text{(S-IFT)}$$

$$\mathbf{if}\ \mathit{false}^l\ \mathbf{then}\ P\ \mathbf{else}\ Q \preceq Q \qquad \text{(S-IFF)}$$

$$*P \preceq *P \mid P \qquad \text{(S-REP)}$$

$$\frac{P \preceq P'}{P \mid Q \preceq P' \mid Q} \qquad \text{(S-PAR)}$$

$$\frac{P \preceq Q}{(\nu x : \xi)\, P \preceq (\nu x : \xi)\, Q} \qquad \text{(S-CNEW)}$$

Definition 4 The reduction relation $\longrightarrow$ is the least relation closed under the following rules:

$$\overline{x}\langle \widetilde{v} \rangle . P \mid x(\widetilde{y}) . Q \longrightarrow P \mid [\widetilde{y} \mapsto \widetilde{v}]Q \qquad \text{(R-COM)}$$

$$\frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R} \quad \text{(R-PAR)}$$

$$\frac{P \longrightarrow Q}{(\nu x : \xi)\, P \longrightarrow (\nu x : \xi)\, Q} \quad \text{(R-NEW)}$$

$$\frac{P \preceq P' \qquad P' \longrightarrow Q' \qquad Q' \preceq Q}{P \longrightarrow Q} \quad \text{(R-SP)}$$

We write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$.

2.3 Examples

As given above, the basic π-calculus has only a few primitives, but various mechanisms present in real programming languages can be easily encoded [24, 30].

We give below some examples of such encodings, which will be used afterwards.[2] Note that our analysis described later does not always give best possible analyses for those encodings; please consult Section 6 for such situations.

For the sake of clarity, we regard integers and operations on them as primitives below.

Example 1 The process $*succ(n, r).\, \overline{r}\langle n+1 \rangle$ works as a function server computing the successor of an integer. It receives a pair consisting of an integer n and a channel r, and sends $n+1$ on channel r. □

Example 2 (locks) A lock (a binary semaphore) can be implemented by using a channel that holds at most one value at any moment. We can regard the presence of a value in the channel as the unlocked state, and the absence of a value as the locked state. Then, creation of a new lock `let x=newlock() in P` is encoded into a process: $(\nu x)\, (\overline{x}\langle\,\rangle \mid P)$. Lock/unlock operations `lock(x);P` and `unlock(x);P` can be encoded into processes $x(\,).\, P$ and $\overline{x}\langle\,\rangle \mid P$ respectively. (Note that the unlock operation is encoded into an asynchronous output since the unlock operation is a non-blocking operation.) □

Example 3 (shared variables) Shared variables can also be implemented by using a channel that holds at most one value at any moment. Creation of a shared variable `let x=newref v in P`, the read operation `let y=!x in P`, and the write operation `x:=v; P` can be encoded respectively into: $(\nu x)\, (\overline{x}\langle v \rangle \mid P)$, $x(y).\, (\overline{x}\langle y \rangle \mid P)$, and $x(y).\, (\overline{x}\langle v \rangle \mid P)$. Another way for encoding shared variables [22] would be to represent a variable as a concurrent object with two methods: *read* and *write*. See Example 7 for how to encode a concurrent object. □

Example 4 (sequencing) A sequential execution `P;Q` can also be easily encoded by using the idea of continuation-passing. Let P be a process that simulates `P` and signals its termination to a channel c, and Q be a process that simulates `Q`. Then, `P;Q` is encoded into $(\nu c)\, (P \mid c(\,).\, Q)$. □

[2] We give below only one encoding for each construct; interested readers can consult other encodings in the literature.

Thread A	**Thread B**	**Thread C**	**Thread D**
`if(!secret)`	`lock(x);`	`lock(y);`	`lock(x);`
`{lock(x);`	`lock(y);`	`lock(x);`	`unlock(x);`
`unlock(x)};`	`unlock(y);`	`unlock(x);`	`public := false`
`public := true`	`unlock(x);`	`unlock(y);`	

Fig. 1 Threads using lock-primitives

Example 5 Let us consider the four threads in Figure 1, which use the above three features (locks, shared variables, and sequencing). Thread A first reads the value of the shared variable `secret`, and if it is true, acquires and releases the lock `x`, and then updates the value of the shared variable `public`. Thread B and C both acquire and release locks `x` and `y`, but in a different order. Thread D acquires and releases the lock `x`, and then updates the value of the shared variable `public`. Here, we assume that the shared variable `public` and the lock `y` can be accessed by anyone (who may be untrusted), while the shared variable `secret` and the lock `x` can be accessed by only privileged principals.

The threads are encoded into the π-calculus as follows.

$$
\begin{aligned}
A &\triangleq (\nu c^{\mathbf{H}})\, secret^{\mathbf{H}}(b^{\mathbf{H}}).\,(\overline{secret^{\mathbf{H}}}\langle b^{\mathbf{H}}\rangle \\
&\quad \mid \mathbf{if}\ b^{\mathbf{H}}\ \mathbf{then}\ x^{\mathbf{H}}().\,(\overline{x^{\mathbf{H}}}\langle\rangle \mid \overline{c^{\mathbf{H}}}\langle\rangle)\ \mathbf{else}\ \overline{c^{\mathbf{H}}}\langle\rangle \\
&\quad \mid c^{\mathbf{H}}().\,public(z)\,.\,\overline{public}\langle \mathit{true}^{\mathbf{L}}\rangle\,) \\
B &\triangleq x^{\mathbf{H}}().\,y^{\mathbf{L}}().\,(\overline{y^{\mathbf{L}}}\langle\rangle \mid \overline{x^{\mathbf{H}}}\langle\rangle) \\
C &\triangleq y^{\mathbf{L}}().\,x^{\mathbf{H}}().\,(\overline{x^{\mathbf{H}}}\langle\rangle \mid \overline{y^{\mathbf{L}}}\langle\rangle) \\
D &\triangleq x^{\mathbf{H}}().\,(\overline{x^{\mathbf{H}}}\langle\rangle \mid public^{\mathbf{L}}(z)\,.\,\overline{public^{\mathbf{L}}}\langle \mathit{false}^{\mathbf{L}}\rangle\,)
\end{aligned}
$$

The channel c in the process A is used to signal the termination of the if-expression. We have annotated variables with their secrecy levels to clarify the assumption.

Note that none of the above processes is well-typed in Pottier's type system [27]; all the processes try to perform communication on low-level channels after receiving values on high-level channels. Some combinations of those processes should, however, still be considered safe (if we ignore timing leaks). For example, $A|C$ and $A|D$ are safe provided that other privileged principals use locks x properly, without holding lock x forever. Although A, C and D perform communication on *public* after communicating over a high-level channel c or x, the communication on c or x always succeeds, so that no secret information is leaked. On the other hand, $A|B$ is (arguably) unsafe, on the assumption that low-level processes may maliciously lock y forever. In that case, Thread B may be blocked forever at the statement `lock(y)` and fail to release the lock x, which would cause A to get blocked only if the value of `secret` is true. Thus, a low-level process can guess the value of `secret` by reading the value of `public`. We will later show that $A|C$ and $A|D$ are indeed well-typed in our type system, while $A|B$ is not. □

Example 6 The **cobegin**/**coend** statement:
`cobegin P1 | ... | Pn coend` (which executes `P1`,..., `Pn` concurrently) can also be easily encoded. Let P_i be a process that simulates `Pi` and

signals its termination to c_i. Then, `cobegin P1 | ... | Pn coend; Q` can be encoded into a process: $(\nu c_1) \cdots (\nu c_n)\,(P_1 \mid \cdots \mid P_n \mid c_1(\,).\cdots c_n(\,).\,Q)$.

This construct can be freely combined with other communication or synchronization primitives. Consider the following program:

```
cobegin
    (x:=true; send(c, nil))
  | let _=receive(c) in y:=!x
coend;
w := false
```

Here, `send` and `receive` are commands to synchronously send/receive values. The first thread of the `cobegin` is a producer, which writes `true` to the variable `x` and notify that it is ready, and the second one waits for the notification and reads the variable `x`. It can be encoded into:

$$\begin{array}{l}(\nu c_1)\,(\nu c_2)\,(x(z).\,(\overline{x}\langle \mathit{true}^{\mathbf{H}}\rangle \mid \overline{c}\langle\,\rangle.\,\overline{c_1}\langle\,\rangle)\\ \quad \mid c(\,).\,x(z).\,(\overline{x}\langle z\rangle \mid y(u).\,(\overline{y}\langle z\rangle \mid \overline{c_2}\langle\,\rangle))\\ \quad \mid c_1(\,).\,c_2(\,).\,w(z)\,.\,\overline{w}\langle \mathit{false}^{\mathbf{L}}\rangle)\end{array}$$

□

Example 7 Concurrent objects can be easily expressed in the π-calculus [20, 26]. The following process implements a bank account object:

$$\begin{array}{l}(\nu s)\,(\overline{s}\langle 100^{\mathbf{H}}\rangle\\ \quad \mid *\mathit{withdraw}^{\mathbf{H}}(\mathit{amount}, r).\,s(x).\\ \qquad\quad \mathbf{if}\ x \geq \mathit{amount}\ \mathbf{then}\ (\overline{r}\langle \mathit{true}^{\mathbf{H}}\rangle \mid \overline{s}\langle x - \mathit{amount}\rangle)\\ \qquad\quad \mathbf{else}\ (\overline{r}\langle \mathit{false}^{\mathbf{H}}\rangle \mid \overline{s}\langle x\rangle)\\ \quad \mid *\mathit{getBalance}^{\mathbf{H}}(r).\,s(x).\,(\overline{r}\langle x\rangle \mid \overline{s}\langle x\rangle)\\ \quad \mid *\mathit{deposit}^{\mathbf{L}}(\mathit{amount}, r).\,s(x).\,(\overline{r}\langle\,\rangle \mid \overline{s}\langle x + \mathit{amount}\rangle))\end{array}$$

Here, we follow Pierce and Turner's encoding of concurrent objects [26], where concurrent objects are realized as a set of processes, each of which realizes each method. The above process stores the current balance in the channel s, and handles three kinds of requests through channels *withdraw*, *getBalance*, and *deposit*. Upon receiving a request on *withdraw*, the process checks whether the current balance is enough and replies whether the withdraw operation has succeeded or not. Upon receiving a request on *getBalance*, the process sends the current balance to the client. Upon receiving a request on *deposit* (for depositing the specified amount of money to this account), the process updates the current balance and sends an acknowledgment on r. The channels *withdraw* and *getBalance* are secret, so that only a privileged person can send requests, while the channel *deposit* (for sending a request for transferring money to this account) can be accessed by anyone. The current balance should also be kept secret, so that 100 is annotated with $\mathbf{H}$.

The type system presented in the next section can guarantee that one cannot obtain any information about the current balance through the public channel *deposit* (see Example 17). Note that the previous type systems for information flow-analysis for the π-calculus and similar calculi [10, 11, 27, 36] cannot accept the process above, since the sub-process $*\mathit{deposit}(\mathit{amount}, r).\,\cdots$ sends a message through a non-secret channel r after receiving a value through the secret channel

s. (One may, however, use other encodings so that a process having the same functionality is well-typed in previous type systems [11]. So, the point here is rather about the synchronization between multiple threads inside the same object; in the above process, synchronization occurs through channel s. The type system should be able to infer that the synchronization succeeds to conclude that information about the current balance is not leaked.)

Channels *withdraw*, *getBalance*, and *deposit* can be passed around through other channels as identities of the bank account object above. The following process stores the channel *withdraw* in the shared variable *y*, reads it, invokes the *withdraw* method, and then waits for a reply on a new channel r (recall the encoding of shared variables in Example 3):

$$
\begin{array}{ll}
y^{\mathbf{H}}(x).\,(\overline{y^{\mathbf{H}}}\langle \mathit{withdraw}\rangle \mid & \text{store } \mathit{withdraw} \text{ in variable } y \\
\quad y^{\mathbf{H}}(m).\,(\overline{y^{\mathbf{H}}}\langle m\rangle \mid & \text{read the channel stored in } y \\
\quad\quad (\nu r)\,\overline{m}\langle 10, r\rangle.\,r(b).\,\mathbf{0})) & \text{invoke the method and wait for a reply}
\end{array}
$$

□

3 Type System for Information Flow Analysis

This section introduces a type system for information flow analysis. The type system guarantees that any well-typed process does not leak secret information, so that the problem of checking whether a process leaks secret information is reduced to the problem of type inference. We first explain main ideas of the type system in Subsection 3.1, and then introduce formal definitions in later subsections.

3.1 Overview

We explain ideas of the type system informally in three steps.

3.1.1 Extending types with secrecy levels

As in other type systems for information flow analysis, we extend the usual types with secrecy levels. For example, the type $\mathbf{bool}$ of booleans is refined to $\mathbf{bool}^{\mathbf{H}}$ and $\mathbf{bool}^{\mathbf{L}}$: The former is the type of secret booleans and the latter is the type of non-secret booleans.

Since information about values may be propagated through the behavior of processes, we also need to consider secrecy levels of *channels* and *processes*. For example, if a process $\mathbf{if}\ \mathit{true}^{\mathbf{H}}\ \mathbf{then}\ \overline{x}\langle\,\rangle\ \mathbf{else}\ \mathbf{0}$ is executed, information about the boolean $\mathit{true}^{\mathbf{H}}$ is propagated through information about whether a message is sent on x or not. Moreover, if a process $x(\,).\,P$ is executed in parallel, the information is further propagated through information about whether the process P is executed. To keep track of this kind of information flow, we let the secrecy level of a channel express the degree of secrecy of information about what communication takes place on the channel (e.g., whether some message is sent on the channel), and let the secrecy level of a process express the degree of secrecy of information about

whether the process is executed. (The latter corresponds to the secrecy level of a program counter in information flow analysis for imperative languages [5].) In the example above, we consider that the secrecy levels of the channel x and the process P are also $\mathbf{H}$.

We write $\langle\tau\rangle^l$ for the type of a channel of secrecy level l that is used for transmitting values of type τ. The secrecy level of a channel should not be confused with the secrecy level of values sent on the channel. For example, if a channel has type $\langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{L}}$, then a non-privileged principal may obtain information about whether some message is sent along the channel although he or she cannot obtain information about the contents of the message.

A type judgment for a process is of the form $x_1:\tau_1,\ldots,x_n:\tau_n \vdash_l P$, where $\tau_1,\ldots,\tau_n$ are extended types and l denotes the secrecy level of the behavior of P. The secrecy level l corresponds to "*pc*" found in analyses for sequential languages [5]. If l is $\mathbf{H}$, the behavior of P is only observable to privileged principals. For example, $x:\langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{H}} \vdash_{\mathbf{H}} \overline{x}\langle \mathit{true}^{\mathbf{H}}\rangle$ is valid, but $x:\langle\mathbf{bool}^{\mathbf{L}}\rangle^{\mathbf{L}} \vdash_{\mathbf{H}} \overline{x}\langle \mathit{true}^{\mathbf{L}}\rangle$ is invalid, since the latter process sends a value on a non-secret channel, so that its behavior is observable to any principals.

Based on the intuition above, it would not be difficult to understand the following rule for if-expressions:

$$\frac{\Gamma \vdash v:\mathbf{bool}^{l_1} \qquad \Gamma \vdash_{l_2} P \qquad \Gamma \vdash_{l_2} Q \qquad l_1 \sqsubseteq l_2}{\Gamma \vdash_{l_2} \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q}$$

Since one can infer the value of v by observing whether P or Q is executed, the secrecy level l_2 of P and Q must be greater than or equal to the secrecy level l_1 of v.

Based on a similar intuition, we can obtain the following rule for an input process $x(y).P$:

$$\frac{\Gamma, x:\langle\tau\rangle^{l_1}, y:\tau \vdash_{l_2} P \qquad l \sqsubseteq l_1 \sqsubseteq l_2}{\Gamma, x:\langle\tau\rangle^{l_1} \vdash_l x(y).P} \qquad \text{(IN-NAIVE)}$$

Since one can infer whether some process sends a value on x by observing whether P is executed or not, the rule imposes the condition $l_1 \sqsubseteq l_2$.

3.1.2 Extending channels types with usage expressions

The rule IN-NAIVE given above is actually too naive. Because of the condition $l_1 \sqsubseteq l_2$, once a process performs an input on a secret channel, it can no longer perform communications on non-secret channels. For example, the process A in Example 5 cannot be typed: Since b is a secret value, both channels x and c must be secret, so that $c(\,).\mathit{public}(z).\overline{\mathit{public}}\langle \mathit{true}^{\mathbf{L}}\rangle$ cannot be typed. Actually, however, as long as the input on x always succeeds, the input on c also succeeds, so that the process does not leak any information about n.

Based on the above observation, we want to replace rule IN-NAIVE with something like:[3]

[3] Actually, additional subtle conditions are required in order for this rule to be valid: See the paragraph on well-formedness conditions in Section 6.

$$\frac{\Gamma, x : \langle\tau\rangle^{l_1}, y : \tau \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad l_1 \sqsubseteq l_2 \text{ if the input on } x \text{ may not succeed}}{\Gamma, x : \langle\tau\rangle^{l_1} \vdash_l x(y).\, P} \qquad \text{(IN-IDEAL)}$$

What remains to do is to replace the sentence "if the input on x may not succeed" with a well-defined and statically verifiable condition. For this purpose, we use the idea of type systems for lock-freedom [14, 18, 32]. The idea is to extend channel types with *usage expressions* (usages, in short) [14, 18, 32], which specify how each channel should be used by each process.

Usages are constructed from I, denoting an input action, and O, denoting an output action, by using sequential composition (.), parallel composition ($\mid$), etc. For example, usage $I.O$ describes a channel that should be first used once for input *and then* used once for output. So, if x has usage $I.O$, the process $x(\,).\,\overline{x}\langle\,\rangle$ is valid but the processes $\overline{x}\langle\,\rangle.\,x(\,)$ and $x(\,).\,(\overline{x}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$ are invalid. Usage $I \mid O$ describes a channel that should be used once for input and once for output *in parallel*. So, if x has usage $I \mid O$, then the process $x(\,) \mid \overline{x}\langle\,\rangle$ is valid. A channel type is annotated with a usage and written $\langle\tau\rangle^l/U$, which describes a channel that is used according to usage U.

Typing rules are extended to take into account usage information by using the idea of linear types [17]. For example, the rule for parallel composition is:

$$\frac{\Gamma_1 \vdash_l P_1 \qquad \Gamma_2 \vdash_l P_2}{\Gamma_1 \mid \Gamma_2 \vdash_l P_1 \mid P_2}$$

Here, $\Gamma_1 \mid \Gamma_2$ is the type environment obtained by combining usages of each variable in Γ_1 and Γ_2 with $\mid$. For example, from $x : \langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{H}}/I \vdash_{\mathbf{H}} P_1$ and $x : \langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{H}}/O \vdash_{\mathbf{H}} P_2$ (which mean that x is used once for input in P_1 and once for output in P_2), we can derive $x : \langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{H}}/(I \mid O) \vdash_{\mathbf{H}} P_1 \mid P_2$, which means that x is used once for input and once for output in parallel. The rule for input processes would be replaced by (for the moment, we forget conditions on secrecy levels and omit them):

$$\frac{\Gamma, x : \langle\tau\rangle/U, y : \tau \vdash P}{\Gamma, x : \langle\tau\rangle/I.U \vdash x(y).\, P} \qquad \text{(IN-WITH-USAGE)}$$

Here, the usage of x in the conclusion captures the fact that x is first used for input and then used according to U in P.

Similarly, the rule for output processes would be:

$$\frac{\Gamma, x : \langle\tau\rangle/U \vdash P}{\Gamma, x : \langle\tau\rangle/O.U, y : \tau \vdash \overline{x}\langle y\rangle} \qquad \text{(OUT-WITH-USAGE)}$$

Using the above rules, we obtain the following type derivation for $x(y).\,\overline{y}\langle 1\rangle \mid \overline{x}\langle z\rangle$:

$$\dfrac{\dfrac{x : \langle\langle int\rangle/O\rangle/0, y : \langle int\rangle/O \vdash \overline{y}\langle 1\rangle}{x : \langle\langle int\rangle/O\rangle/I \vdash x(y).\,\overline{y}\langle 1\rangle} \qquad x : \langle\langle int\rangle/O\rangle/O, z : \langle int\rangle/O \vdash \overline{x}\langle z\rangle}{x : \langle\langle int\rangle/O\rangle/(I|O), z : \langle int\rangle/O \vdash x(y).\,\overline{y}\langle 1\rangle \mid \overline{x}\langle z\rangle}$$

From the conclusion, we can deduce that x is used once for input and output, and z is used once for output.

From the usage part of a channel type, we can obtain some information about whether communications succeed or not. For example, in the process $(\nu x)\, x(\,).\, \overline{y}\langle\,\rangle$, the usage of x is expressed by I. So, we know that x is used once for input but never used for output, so that the input never succeeds.

3.1.3 Refining usages with obligation/capability levels

Usage expressions explained above are not sufficient for the purpose of checking that certain communications *must* succeed, since they can express only channel-wise communication behavior of processes, not inter-channel dependencies. For example consider the process $x(\,).\, \overline{y}\langle\,\rangle \mid y(\,).\, \overline{x}\langle\,\rangle$. Usage of x and y can be expressed by $I \mid O$, but communications on x and y never succeed because of a deadlock. The problem of the above process can be explained as follows: In order for the input on x to succeed, the righthand process has an *obligation* to do an output on x. Before fulfilling the obligation, however, the righthand process is claiming a *capability* to successfully complete the input on y. In order for the input on y to succeed, the lefthand process has an *obligation* to do an output on y, but before fulfilling the obligation, the lefthand process is claiming a capability to successfully complete the output on x. Thus, both processes claim their capabilities before fulfilling the obligations, so that a deadlock occurs.

In order to avoid this kind of circular dependency, we associate each I and O in usages with an *obligation level* t_o and a *capability level* t_c, and write $I^{t_o}_{t_c}$ and $O^{t_o}_{t_c}$. Obligation levels and capability levels range over the set consisting of natural numbers and ∞. The obligation level expresses the degree of an obligation to do an action, while the capability level expresses the degree of a capability to successfully complete an action. More precisely, obligation and capability levels control the behavior of processes through the following rules:

Rule A If a process holds an obligation of level n to do some action, then the process can exercise only a finite number of capabilities whose levels are less than n before fulfilling the obligation. As a special case, if a process holds an obligation of level 0 to do some action, then the process must do the action immediately; no prefix is allowed. On the other hand, if a process holds an obligation of level ∞, then the process need not do the action at all. For example, if the usage of x is O^2_3 and the usage of y is O^2_0, the process $\overline{y}\langle\,\rangle.\, \overline{x}\langle\,\rangle$ is allowed since the process tries to use the capability of level 0 to send a value on y before fulfilling the obligation of level 1 to send a value on x. On the other hand, $\overline{x}\langle\,\rangle.\, \overline{y}\langle\,\rangle$ is not allowed since the process tries to use the capability of level 3 to send a value on x before fulfilling the obligation of level 2, which is less than 3.

Rule B If a process holds a capability of level n to perform an action, there must exist another process that holds an obligation to do its co-action whose level is less than or equal to n. For example, if a process holds a channel of usage O^∞_0, then there must be another process that uses the channel according to I^0_n (where n can be any level), so that the capability of level 0 to send a value by the former process is guaranteed by the second process's obligation to receive the value.

These rules ensure that every action of a finite capability level will eventually succeed. For example, consider the process $x(\,).\,\overline{y}\langle\,\rangle \mid y(\,) \mid \overline{x}\langle\,\rangle$. We can assign usages $I_0^0 \mid O_0^0$ and $I_1^0 \mid O_0^1$ to x and y, so that the above rules are satisfied. Since the capability levels are finite, we can conclude that the communications on x and y succeed.

On the other hand, for the deadlocked process $x(\,).\,\overline{y}\langle\,\rangle \mid y(\,).\,\overline{x}\langle\,\rangle$, there is no way to assign a finite capability level to the input on x: Let usages of x and y be $I_{t_2}^{t_1} \mid O_{t_4}^{t_3}$ and $I_{t_6}^{t_5} \mid O_{t_8}^{t_7}$. Rule A requires $t_2 < t_7$ and $t_6 < t_3$. On the other hand, Rule B requires the following conditions:

$$t_1 \leq t_4 \qquad t_3 \leq t_2 \qquad t_5 \leq t_8 \qquad t_7 \leq t_6.$$

So, we obtain the constraint $t_2 < t_7 \leq t_6 < t_3 \leq t_2$, so that the capability level t_2 cannot be finite.

The typing rule for input processes is now refined as follows.

$$\frac{\Gamma, x : \langle\tau\rangle^{l_1}/U, y : \tau \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\uparrow^{(t_c+1,t_c+1)}\Gamma, x : \langle\tau\rangle^{l_1}/I_{t_c}^{t_o}.U \vdash_l x(y).\,P} \quad \text{(IN-REFINED)}$$

$\uparrow^{(t_c+1,t_c+1)}\Gamma$ lifts the obligations whose levels are less than or equal to t_c up to $t_c + 1$, to enforce Rule A. The statement "if the input on x may not succeed" has now been replaced by a statically verifiable condition $t_c = \infty$.

Rule B is enforced by the following typing rule for ν-prefix:

$$\frac{\Gamma, x : \langle\tau\rangle^{l_1}/U \vdash_{l_2} P \qquad rel(U)}{\Gamma \vdash_{l_2} (\nu x)\,P} \quad \text{(NEW)}$$

The condition $rel(U)$ means that for any usage of the form $I_{t_c}^{t_o}.U_1$ in U where t_c is finite, there is a corresponding usage of the form $O_{t_c}^{t'_o}.U_2$ such that $t'_o \leq t_c$ (and a similar condition for $O_{t_c}^{t_o}.U_1$ in U) and that the same condition must hold for any U' obtained by discharging a matching I and O in U. For example, in order for $rel(I_{t_2}^{t_1} \mid O_{t_4}^{t_3})$ to hold, it must be the case that $t_1 \leq t_4$ and $t_3 \leq t_2$. $rel(O_\infty^0 \mid I_0^\infty.O_\infty^0)$ holds but $rel(O_\infty^0 \mid I_0^\infty.O_0^0)$ does not, since discharging a pair of I and O in the latter usage yields O_0^0, which has a finite capability level but there is no corresponding usage of the form $I_n^0.U$.

A part of the requirement of Rule A, that only a *finite* number of capabilities can be exercised before an obligation is fulfilled, is enforced by the following rule for output processes (we omit the continuation part for the sake of simplicity):

$$\frac{l \sqsubseteq l_1}{x : \langle\tau\rangle^{l_1}/O_{t_c}^{t_o}.U, y : \uparrow^{(t_c+1,t_c+1)}\uparrow\tau \vdash_l \overline{x}\langle y\rangle} \quad \text{(OUT)}$$

Like in IN-REFINED, we apply $\uparrow^{(t_c+1,t_c+1)}$ to the type of x to enforce Rule A. In addition, we apply an operator $\uparrow$ to increase obligation levels of τ by one. For example, $\uparrow(\langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{H}}/O_2^1) = \langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{H}}/O_2^2$. This enforces that when an obligation is delegated to another process (by sending it through a channel), the level of the delegated obligation is less than the current obligation level, so that

Usages	Interpretation
$\mathbf{0}$	Cannot be used at all
$I^{t_o}_{t_c}.U$	Used once for input, and then used according to U
$O^{t_o}_{t_c}.U$	Used once for output, and then used according to U
$U_1 \mid U_2$	Used according to U_1 and U_2, possibly in parallel
$*U$	Used according to U by infinitely many processes
$\uparrow^{(t_1,t_2)}U$	The same as U, except that input and output obligation levels are raised to t_1 and t_2 respectively.
$U_1 \,\&\, U_2$	Used according to either U_1 or U_2
ρ	Usage variable (used in combination with recursive usages below)
$\mu\rho.U$	Recursively used according to $[\rho \mapsto \mu\rho.U]U$.

Table 1 Meaning of Usage Expressions

an obligation of a finite level cannot be infinitely delegated (since the level of the obligation eventually reaches 0). For example, consider the process

$\overline{x_1}\langle y\rangle \mid x_1(z).\,\overline{x_2}\langle z\rangle \mid x_2(z).\,P.$

Suppose that the sub-process $\overline{x_1}\langle y\rangle$ initially holds an obligation of level 2 to do an output on y. When y is sent through x_1, the obligation level becomes 1, and when it is further sent through x_2, the obligation level becomes 0, so that the process P must use y immediately. In the process $*x(z).\,\overline{x}\langle z\rangle \mid \overline{x}\langle y\rangle$, which forwards y forever, y's obligation level must be ∞.

3.2 Usages

3.2.1 Syntax

Now we introduce the formal syntax of usages.

Definition 5 (usages) The set $\mathcal{U}$ of *usages*, ranged over by U, is given by the following syntax.

$U ::= \mathbf{0} \mid \alpha^{t_1}_{t_2}.U \mid (U_1 \mid U_2) \mid *U \mid \uparrow^{(t_1,t_2)}U \mid U_1 \,\&\, U_2 \mid \rho \mid \mu\rho.U$
$\alpha ::= I \mid O$
$t_1, t_2 \in \mathbf{Nat} \cup \{\infty\}$

Here, $\mathbf{Nat}$ is the set of natural numbers.

We often omit $\mathbf{0}$ and write $\alpha^{t_1}_{t_2}$ for $\alpha^{t_1}_{t_2}.\mathbf{0}$. We extend the usual binary relation $\leq$ on $\mathbf{Nat}$ to that on $\mathbf{Nat} \cup \{\infty\}$ by $\forall t \in \mathbf{Nat} \cup \{\infty\}.t \leq \infty$. We write $\mathbf{min}(x_1, \ldots, x_n)$ for the least element of $\{x_1, \ldots, x_n\}$ (∞ if $n = 0$) with respect to $\leq$ and write $\mathbf{max}(x_1, \ldots, x_n)$ for the greatest element of $\{x_1, \ldots, x_n\}$ (0 if $n = 0$). We assume that $\mu\rho$ binds ρ. We write $[\rho \mapsto U_1]U_2$ for the usage obtained by replacing the free occurrences of ρ in U_2 with U_1. We write $FV(U)$ for the set of free usage variables. A usage is *closed* if $FV(U) = \emptyset$.

Intuitive meaning of usages is summarized in Table 1. Additional explanation is in order. If t_o is finite, a channel of usage $\alpha^{t_o}_{t_c}.U$ must be used for the action α, while if t_o is ∞, the channel need not be used. If t_c is finite, whenever the action α

is tried on a channel of usage $\alpha^{t_o}_{t_c}.U$, the action will eventually succeed. If t_c is ∞, there is no such guarantee. A channel of usage $\alpha^{t_o}_{t_c}.U$ must be used according to U only when it has been used for the action α and the action succeeds. For example, a channel of usage $I^{\infty}_0.O^0_{\infty}$ can be used for input (but need not be used), and if it has been used for input and the input has succeeded, it *must* be used for output. Usage $\uparrow^{(t_1,t_2)}U$ lifts the obligation levels occurring in U (except for those guarded by I or O) so that the input obligations and output obligations become greater than or equal to t_1 and t_2 respectively. For example, $\uparrow^{(1,2)}(I^0_0.O^0_{\infty} \mid O^3_0 \mid O^0_3)$ is the same as $I^1_0.O^0_{\infty} \mid O^3_0 \mid O^2_3$.[4]

Choice $U_1 \,\&\, U_2$ and recursive usages $\mu\rho.U$ are only required to enable type inference [18], so that they can be skipped at first reading. Usage $\mu\rho.O^{\infty}_0.\rho$ describes a channel that can be used for output infinitely often.[5]

Notation 2 We give a higher precedence to prefixes ($\alpha^{t_o}_{t_c}$ and $*$) than to $\mid$. So, $I^{t_o}_{t_c}.U_1 \mid U_2$ means $(I^{t_o}_{t_c}.U_1) \mid U_2$, not $I^{t_o}_{t_c}.(U_1 \mid U_2)$. We write $\overline{\alpha}$ for the co-action of α ($\overline{I} = O$ and $\overline{O} = I$).

Example 8 Linear channels (channels that are used once for input and once for output) [17] are given a usage of the form $I^{t_1}_{t_2} \mid O^{t_3}_{t_4}$. For example, the channel c in Example 4 is given a usage $I^1_1 \mid O^1_1$. The usage of channels used for client-server connection (like $succ$ in Example 1) is expressed as $*I^0_{\infty} \mid *O^{\infty}_0$. The part $*I^0_{\infty}$ means that a server must wait for requests forever, and the part $*O^{\infty}_0$ means that clients can send an infinite number of requests, and that it is guaranteed that the requests are received by a server.

Example 9 The usage of a lock channel (i.e., a channel used as a lock: recall Example 2) is expressed by $O^n_{\infty} \mid *I^{\infty}_n.O^n_{\infty}$ for some $n \in \mathbf{Nat}$. The part O^n_{∞} says that a value must be first put into the channel (to initialize the lock), and the part $I^{\infty}_n.O^n_{\infty}$ says that the lock can be eventually acquired, and after the lock has been acquired, then the lock must be released. The natural number n can be used to control in which order locks are acquired. Suppose that lock channels x and y have usages $*I^{\infty}_{n_x}.O^{n_x}_{\infty}$ and $*I^{\infty}_{n_y}.O^{n_y}_{\infty}$ respectively and $n_y < n_x$ holds. Then, the process $x().y().(\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)$, which locks x and y in this order, is allowed, but the process $y().x().(\overline{x}\langle\rangle \mid \overline{y}\langle\rangle)$ is not allowed. Note that the latter process tries to exercise the capability of level n_x to lock x before fulfilling the obligation of level n_y to release the lock y. □

Example 10 The usage of a channel implementing a shared variable (recall Example 3) is expressed as $O^0_{\infty} \mid *I^{\infty}_0.O^0_{\infty}$. It is a special case of the usage of lock channels. The capability level of an input action being 0 captures the fact that read/write operations always succeed, and the obligation level of an output action

[4] Readers who are familiar with our previous type system for lock-freedom[14] find $\uparrow^{(t_1,t_2)}U$ to be similar to $\boxed{\mathsf{t}}U$. The readers should not, however confuse between them. While $\boxed{\mathsf{t}}O^{t'}_{\infty}$ is $O^{t+t'}_{\infty}$, $\uparrow^{(t,t)}O^{t'}_{\infty}$ expresses $O^{\mathbf{max}(t,t')}_{\infty}$. Please also note that $\boxed{\mathsf{t}}U$ was an operation, while $\uparrow^{(t_1,t_2)}$ is a constructor. $\uparrow^{(t_1,t_2)}$ cannot be defined as an operation because of the presence of usage variables.

[5] A careful reader may think that $*U$ can be represented by $\mu\rho.(\rho \mid U)$. The formal semantics of $*U$ and $\mu\rho.(\rho \mid U)$, given later, are different in a subtle way.

being 0 captures the fact that a value must be immediately written back when a value is extracted from the channel. □

Example 11 (choice and recursion)
The process $\mathbf{if}\ b\ \mathbf{then}\ \overline{x}\langle\,\rangle\ \mathbf{else}\ (\overline{x}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$ uses the channel x according to $O^0_\infty \,\&\, (O^0_\infty \mid O^0_\infty)$. The usage $\mu\rho.O^\infty_\infty.\rho$ describes channels that can be used for output repeatedly. For example, the process $\overline{\mathit{repeat}}\langle x\rangle \mid *\mathit{repeat}(y).\,\overline{y}\langle\,\rangle.\,\overline{\mathit{repeat}}\langle y\rangle$ uses the channel x according to that usage. □

3.2.2 Semantics

The formal meaning of a usage is determined by its obligation/capability levels of a usage, which represent what capabilities/obligations currently exist, and its reduction, which expresses how the usage changes during reduction of processes.

We first define capability/obligation levels of a usage.

Definition 6 (capabilities) The *input and output capability levels* of usage U, written $cap_I(U)$ and $cap_O(U)$, are defined by:

$$
\begin{array}{l}
cap_\alpha(\mathbf{0}) = cap_\alpha(\overline{\alpha}^{t_o}_{t_c}.U) = cap_\alpha(\rho) = \infty \\
cap_\alpha(\alpha^{t_o}_{t_c}.U) = t_c \\
cap_\alpha(*U) = cap_\alpha(\uparrow^{(t_1,t_2)}U) = cap_\alpha(\mu\rho.U) = cap_\alpha(U) \\
cap_\alpha(U_1 \mid U_2) = cap_\alpha(U_1 \,\&\, U_2) = \mathbf{min}(cap_\alpha(U_1), cap_\alpha(U_2))
\end{array}
$$

Definition 7 (obligations) The *input and output obligation levels* of a closed usage U, written $ob_I(U)$ and $ob_O(U)$, are defined by:

$$
\begin{array}{l}
ob_\alpha(U) = ob^\emptyset_\alpha(U) \\
ob^F_\alpha(\mathbf{0}) = ob^F_\alpha(\overline{\alpha}^{t_o}_{t_c}.U) = \infty \\
ob^F_\alpha(\rho) = F(\rho) \\
ob^F_\alpha(\alpha^{t_o}_{t_c}.U) = t_o \\
ob^F_\alpha(U_1 \mid U_2) = \mathbf{min}(ob^F_\alpha(U_1), ob^F_\alpha(U_2)) \\
ob^F_\alpha(\uparrow^{(t_I,t_O)}U) = \mathbf{max}(t_\alpha, ob^F_\alpha(U)) \\
ob^F_\alpha(U_1 \,\&\, U_2) = \mathbf{max}(ob^F_\alpha(U_1), ob^F_\alpha(U_2)) \\
ob^F_\alpha(*U) = ob^F_\alpha(U) \\
ob^F_\alpha(\mu\rho.U) = \mathbf{lfp}(\lambda x.ob^{F[\rho\mapsto x]}_\alpha(U))
\end{array}
$$

Here, $\mathbf{lfp}$ denotes the least fixed-point operator. We write $ob(U)$ for $\mathbf{min}(ob_I(U), ob_O(U))$.

The above definition uses a sub-function ob^F_α, where F is a mapping from usage variables to obligation levels. For example, $ob_O(\mu\rho.(\rho \mid O^1_2)) = \mathbf{lfp}(\lambda x.\mathbf{min}(x,1)) = 0$. On the other hand, $ob_O(*O^1_2) = 1$. (So, $*U$ is not the same as $\mu\rho.(\rho \mid U)$.)

The following lemma guarantees that $ob_\alpha(U)$ defined for any U.

Lemma 1 *Suppose that f is a function from $\mathbf{Nat} \cup \{\infty\}$ to $\mathbf{Nat} \cup \{\infty\}$ and that f is monotonic with respect to $\leq$. Then, f has a least fixed-point.*

Proof The monotonicity of f implies that $0, f(0), f(f(0)), ..., f^n(0), \ldots$ is an increasing sequence. If $f^n(0) = f^{(n+1)}$ holds for some n, $f^n(0)$ is the least fixed-point of f. Otherwise, $0, f(0), f(f(0)), ..., f^n(0), \ldots$ is unbounded. Since f is monotonic, $f^n(0) \leq f(\infty)$ for any n, which implies $f(\infty) = \infty$. Therefore, ∞ is the least fixed-point. □

The usage of a channel describes how the channel should be used afterwards, so the usage changes during reduction of processes. For example, if x has usage $I^{\infty}_{\infty}.O^0_0 \mid O^{\infty}_{\infty}.I^0_0$, after a communication on x occurs, x should be used according to $O^0_0 \mid I^0_0$. This change of usage is expressed by the usage reduction relation defined below. Intuitively, $U \longrightarrow U'$ means that if a channel of usage U has been used for a communication, then it should be used according to U' afterwards. As in the definition of the process reduction relation, we use a structural relation $\preceq$ as an auxiliary relation.

Definition 8 $\preceq$ is the least reflexive and transitive relation on usages satisfying the following rules:

$$U_1 \mid U_2 \preceq U_2 \mid U_1 \qquad \text{(UP-COMMUT)}$$

$$(U_1 \mid U_2) \mid U_3 \preceq U_1 \mid (U_2 \mid U_3) \qquad \text{(UP-ASSOC)}$$

$$\frac{U_1 \preceq U_1' \qquad U_2 \preceq U_2'}{U_1 \mid U_2 \preceq U_1' \mid U_2'} \qquad \text{(UP-CONGP)}$$

$$*U \preceq *U \mid U \qquad \text{(UP-REP)}$$

$$\uparrow^{(t_I,t_O)} \alpha^{t_1}_{t_2}.U \preceq \alpha^{\mathbf{max}(t_1,t_\alpha)}_{t_2}.U \qquad \text{(UP-}\uparrow\text{)}$$

$$\uparrow^{(t_1,t_2)}(U_1 \mid U_2) \preceq (\uparrow^{(t_1,t_2)} U_1) \mid (\uparrow^{(t_1,t_2)} U_2) \qquad \text{(UP-DIST)}$$

$$U_1 \,\&\, U_2 \preceq U_i \ (i \in \{1,2\}) \qquad \text{(UP-OR)}$$

$$\mu\rho.U \preceq [\rho \mapsto \mu\rho.U]U \qquad \text{(UP-REC)}$$

$$\frac{U \preceq U'}{\uparrow^{(t_1,t_2)} U \preceq \uparrow^{(t_1,t_2)} U'} \qquad \text{(UP-CONG}\uparrow\text{)}$$

Definition 9 (usage reduction) The binary relation $\longrightarrow$ on usages is the least relation closed under the following rules:

$$I^{t_o}_{t_c}.U_1 \mid O^{t_o'}_{t_c'}.U_2 \longrightarrow U_1 \mid U_2$$

$$\frac{U_1 \longrightarrow U_1'}{U_1 \mid U_2 \longrightarrow U_1' \mid U_2}$$

$$\frac{U_1 \preceq U_1' \qquad U_1' \longrightarrow U_2' \qquad U_2' \preceq U_2}{U_1 \longrightarrow U_2}$$

3.2.3 Relations and operations on usages

The relation $rel(U)$ explained in Section 3.1 is formally defined below. It ensures that whenever there is a capability of level n to successfully perform some action, there exists an obligation of the same or lower level to do its co-action.

Definition 10 (reliability) We write $\mathit{con}_\alpha(U)$ when $ob_{\overline{\alpha}}(U) \leq cap_\alpha(U)$. We write $\mathit{con}(U)$ when both $\mathit{con}_I(U)$ and $\mathit{con}_O(U)$ hold. A usage U is *reliable*, written $rel(U)$, if $\mathit{con}(U')$ holds for any U' such that $U \longrightarrow^* U'$.

For example, $rel(O^1_\infty \mid *I^\infty_1.O^1_\infty)$ holds but $rel(O^1_\infty \mid *I^\infty_1.O^2_\infty)$ does not. The latter usage is reduced to $O^2_\infty \mid *I^\infty_1.O^2_\infty$, of which the input capability level is 1 but the output obligation level is 2.

The subusage relation $U_1 \leq U_2$ defined below means that U_1 expresses more liberal usage of channels than U_2, so that a channel of usage U_1 may be used as that of usage U_2. For example, $U_1 \,\&\, U_2 \leq U_1$ and $I^2_2 \leq I^1_3$ hold. (The latter comes from the intuition that an obligation can be replaced by a stronger one, while a capability can be replaced by a weaker one.) We define the subusage relation co-inductively, by using the idea of process simulation relations.

Definition 11 (subusage) The *subusage relation* $\leq$ on closed usages is the largest binary relation on usages such that the following conditions hold whenever $U_1 \leq U_2$.

1. $[\rho \mapsto U_1]U \leq [\rho \mapsto U_2]U$ for any usage U such that $FV(U) = \{\rho\}$.
2. If $U_2 \longrightarrow U'_2$, then there exists U'_1 such that $U_1 \longrightarrow U'_1$ and $U'_1 \leq U'_2$.
3. For each $\alpha \in \{I, O\}$, $cap_\alpha(U_1) \leq cap_\alpha(U_2)$ holds
4. For each $\alpha \in \{I, O\}$, if $\mathit{con}_{\overline{\alpha}}(U_1)$, then $ob_\alpha(U_1) \geq ob_\alpha(U_2)$.

The first and second conditions require that the subusage relation is closed under contexts and reduction. The third condition disallows capabilities to be strengthened, while the fourth condition disallows obligations to be weakened when $\mathit{con}_{\overline{\alpha}}(U_1)$ holds.

We first prove some important properties of the subusage relation.

Lemma 2 *1. If $U_1 \leq U_2$ and $\mathit{con}_\alpha(U_1)$, then $\mathit{con}_\alpha(U_2)$.*
2. If $U_1 \leq U_2$ and $rel(U_1)$, then $rel(U_2)$.
3. The subusage relation $\leq$ is reflexive and transitive.

Proof The first property immediately follow from the assumptions and the third and fourth conditions of the subusage relation: $ob_{\overline{\alpha}}(U_2) \leq ob_{\overline{\alpha}}(U_1) \leq cap_\alpha(U_1) \leq cap_\alpha(U_2)$.

To show the second property, suppose $U_1 \leq U_2$, $rel(U_1)$, and $U_2 \longrightarrow^* U'_2$. By the second condition of Definition 11, there exists U'_1 such that $U_1 \longrightarrow^* U'_1$ and $U'_1 \leq U'_2$. The assumption $rel(U_1)$ implies $\mathit{con}(U'_1)$, from which we obtain $\mathit{con}(U'_2)$ by using the first property of this lemma.

The last property follows from the fact that the identity relation and the composition of $\leq$ satisfies all the conditions of Definition 11. The transitivity of the fourth condition of Definition 11 follows from the first property of this lemma. □

We list some useful laws about the subusage relation. These are part of laws given in Lemma 11 in Appendix A.

Lemma 3 *1. If $U \preceq U'$, then $U \leq U'$.*
2. If $ob(U) = \infty$, then $U \leq \mathbf{0}$.
3. If $t'_o \leq t_o$ and $t_c \leq t'_c$, then $\alpha^{t_o}_{t_c}.U \leq \alpha^{t'_o}_{t'_c}.U$.
4. If $U_1 \leq [\rho \mapsto U_1]U$, then $U_1 \leq \mu\rho.U$.

$\uparrow U$ defined below is the usage obtained by increasing the input and obligation levels of U by one. As explained in Section 3.1, it will be used in the typing rule for output processes.

Definition 12 The operation $\uparrow$ on usages is defined by: $\uparrow U = \uparrow^{(t_1+1,t_2+1)}(U)$ where $t_1 = ob_I(U)$ and $t_2 = ob_O(U)$.

For example, $\uparrow(O^1_0 \mid O^0_1) = \uparrow^{(\infty,1)}(O^1_0 \mid O^0_1)$, which is equivalent to $O^1_0 \mid O^1_1$. (Formally, $\uparrow(O^1_0 \mid O^0_1) \leq O^1_0 \mid O^1_1$ and $O^1_0 \mid O^1_1 \leq \uparrow(O^1_0 \mid O^0_1)$.)

3.3 Types

Definition 13 (types) The set of *types* is given by:

τ (types) $::= \mathbf{bool}^l \mid \mathbf{unit} \mid \xi/U$
ξ (core channel types) $::= \langle \tau_1, \ldots, \tau_n \rangle^l$

Type $\mathbf{bool}^l$ is the type of booleans whose secrecy level is l. Type $\mathbf{unit}$ is the type of the unit value $\star$. A channel type $\langle \widetilde{\tau} \rangle^l / U$ describes a channel that have secrecy level l and should be used according to U for communicating tuples of values of types $\widetilde{\tau}$.

Throughout this paper, we assume that channel types always satisfy the following well-formedness conditions.

Definition 14 A channel type $\langle \widetilde{\tau} \rangle^l / U$ is *well-formed* if it satisfies the following two conditions:

- If $l = \mathbf{H}$, then all the secrecy annotations in $\widetilde{\tau}$ are $\mathbf{H}$.
- If $l = \mathbf{L}$, then all the capability level annotations in U are ∞.

The well-formedness condition allows $\langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{L}} / O^0_\infty$ but neither $\langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{H}} / O^0_\infty$ nor $\langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}} / O^0_0$: the second usage violates the first condition while the third usage violates the second condition. The well-formedness conditions are introduced to simplify the typing rules and the proof of type soundness; how to remove the above conditions is discussed in Section 6. We think that the first condition above is not too restrictive in the presence of subtyping based on secrecy levels (see Section 6). Although the first condition rules out a channel of type $\langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{H}} / O^0_\infty$, it is often (not always, however) the case that one can instead assign $\langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^0_\infty$ to the channel, and coerce a low-level boolean to a high-level boolean before sending it. The second condition is reasonable, since, for the purpose of information flow analysis, analyzing whether communications succeed is important only for secret channels.

We extend relations and operations on usages to those on types.

Definition 15 (subtyping) A subtyping relation $\leq$ is the least reflexive relation closed under the following rule:

$$\frac{U \leq U'}{\xi/U \leq \xi/U'} \qquad \text{(SUBT-CHAN)}$$

For the sake of simplicity, we do not consider subtyping based on secrecy levels (e.g. $\mathbf{bool}^{\mathbf{L}} \leq \mathbf{bool}^{\mathbf{H}}$) and input/output modes [25]. Extension to allow such subtyping is discussed in Section 6.

Definition 16 The *obligation level* of type τ, written $ob(\tau)$, is defined by: $ob(\mathbf{unit}) = ob(\mathbf{bool}^l) = \infty$ and $ob(\xi/U) = ob(U)$.

Definition 17 Unary operations $*$ and $\uparrow$ on types is defined by: $\uparrow\mathbf{unit} = \mathbf{unit}$, $\uparrow\mathbf{bool}^l = \mathbf{bool}^l$, $\uparrow(\xi/U) = \xi/\uparrow U$, $*\mathbf{unit} = \mathbf{unit}$, $*\mathbf{bool}^l = \mathbf{bool}^l$, and $*(\xi/U) = \xi/*U$.

Definition 18 A (partial) binary operation $\mid$ on types is defined by: $\mathbf{unit} \mid \mathbf{unit} = \mathbf{unit}$, $\mathbf{bool}^l \mid \mathbf{bool}^l = \mathbf{bool}^l$, and $(\xi/U_1) \mid (\xi/U_2) = \xi/(U_1 \mid U_2)$. $\tau_1 \mid \tau_2$ is undefined if it does not match any of the above rules.

3.4 Type Environment

A type environment is a mapping from a finite set of variables to types. We use metavariables Γ and Δ for type environments. We write $\emptyset$ for the type environment whose domain is empty. When $x \notin dom(\Gamma)$, we write $\Gamma, x:\tau$ for the type environment Γ' such that $dom(\Gamma') = dom(\Gamma) \cup \{x\}$, $\Gamma'(x) = \tau$, and $\Gamma'(y) = \Gamma(y)$ for all $y \in dom(\Gamma)$.

A type environment Γ is extended to a mapping from a finite set of variables and constants to types, by $\Gamma(\mathit{true}^l) = \Gamma(\mathit{false}^l) = \mathbf{bool}^l$ and $\Gamma(\star) = \mathbf{unit}$. When v is a constant, $\Gamma, v:\tau$ is defined to be Γ only when $\Gamma(v) = \tau$. We abbreviate $\emptyset, v_1:\tau_1, \ldots, v_n:\tau_n$ to $v_1:\tau_1, \ldots, v_n:\tau_n$.

The operations and relations on types are pointwise extended to those on type environments below.

The subtyping relation is extended to a relation on type environments. $\Gamma_1 \leq \Gamma_2$ means that Γ_1 represents more liberal usage of free variables than Γ_2.

Definition 19 A binary relation $\leq$ on type environments is defined by: $\Gamma_1 \leq \Gamma_2$ if and only if (i) $dom(\Gamma_1) \supseteq dom(\Gamma_2)$, (ii) $\Gamma_1(x) \leq \Gamma_2(x)$ for each $x \in dom(\Gamma_2)$, and (iii) $ob(\Gamma_1(x)) = \infty$ for each $x \in dom(\Gamma_1) \backslash dom(\Gamma_2)$.

Definition 20 The operations $\mid$ and $*$ on type environments are defined by:

$$(\Gamma_1 \mid \Gamma_2)(x) = \begin{cases} \Gamma_1(x) \mid \Gamma_2(x) & \text{if } x \in dom(\Gamma_1) \cap dom(\Gamma_2) \\ \Gamma_1(x) & \text{if } x \in dom(\Gamma_1) \backslash dom(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \in dom(\Gamma_2) \backslash dom(\Gamma_1) \end{cases}$$

$$(*\Gamma)(x) = *(\Gamma(x))$$

$$\emptyset \vdash_l \mathbf{0} \quad \text{(T-ZERO)} \qquad \frac{\Gamma, x:\xi/U \vdash_l P \qquad rel(U)}{\Gamma \vdash_l (\nu x:\xi)\, P} \quad \text{(T-NEW)}$$

$$\frac{\Gamma, x:\langle\widetilde{\tau}\rangle^{l_1}/U \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\uparrow^{(t_c+1,t_c+1)}(\Gamma \mid \widetilde{v}:\uparrow\widetilde{\tau}) \mid x:\langle\widetilde{\tau}\rangle^{l_1}/O^0_{t_c}.U \vdash_l \overline{x}\langle\widetilde{v}\rangle.\, P} \quad \text{(T-OUT)}$$

$$\frac{\Gamma \vdash_l P}{*\Gamma \vdash_l *P} \quad \text{(T-REP)}$$

$$\frac{\Gamma \vdash_l P \qquad \Gamma \vdash_l Q}{\Gamma \mid v:\mathbf{bool}^l \vdash_l \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q} \quad \text{(T-IF)}$$

$$\frac{\Gamma, x:\langle\widetilde{\tau}\rangle^{l_1}/U, \widetilde{y}:\widetilde{\tau} \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\uparrow^{(t_c+1,t_c+1)}\Gamma, x:\langle\widetilde{\tau}\rangle^{l_1}/I^0_{t_c}.U \vdash_l x(\widetilde{y}).\, P} \quad \text{(T-IN)}$$

$$\frac{\Gamma' \vdash_{l'} P \qquad \Gamma \leq \Gamma' \qquad l \sqsubseteq l'}{\Gamma \vdash_l P} \quad \text{(T-WEAK)}$$

$$\frac{\Gamma_1 \vdash_l P_1 \qquad \Gamma_2 \vdash_l P_2}{\Gamma_1 \mid \Gamma_2 \vdash_l P_1 \mid P_2} \quad \text{(T-PAR)}$$

Fig. 2 Typing Rules

3.5 Typing Rules

A type judgment is of the form $\Gamma \vdash_l P$, which should be read "P is well typed under Γ and has secrecy level l." It means that P uses free variables as specified by Γ, and the secrecy level of information about its behavior is l. The typing rules for deriving valid type judgments are given in Figure 2. In the rules, "$\Gamma, \widetilde{x}:\widetilde{\tau}$" and "$\Gamma \mid \widetilde{x}:\widetilde{\tau}$" are abbreviations for "$\Gamma, x_1:\tau_1, \ldots, x_n:\tau_n$" and "$\Gamma \mid x_1:\tau_1 \mid \cdots \mid x_n:\tau_n$" respectively. $\widetilde{\tau}' \leq \widetilde{\tau}$ is an abbreviation for $\tau'_1 \leq \tau_1, \ldots, \tau'_n \leq \tau_n$. We explain some rules below. Rules T-IN, T-PAR, and T-NEW have already been explained in Section 3.1.

T-OUT: The operation $\uparrow^{(t_c+1,t_c+1)}$ enforces Rule A given in Section 3.1. It ensures that the level of obligations held by the output process is greater than the level of the capability of the output on x being used. Note that the types of $\widetilde{v}$ in the type environment of the conclusion must be subtypes of $\uparrow\widetilde{\tau}$ rather than $\widetilde{\tau}$: As explained in Section 3.1, it prevents infinite delegations of obligations. The obligation level of the output on x is 0, since it is fulfilled immediately.

T-IF: In combination with T-WEAK, the rule ensures that the secrecy level of the boolean is less than or equal to the secrecy levels of the then-part and the else-part. Note that the type environment $\Gamma \mid v:\mathbf{bool}^l$ in the conclusion implicitly assumes that $\Gamma \mid v:\mathbf{bool}^l$ is well defined; so, the process cannot be typed if $v = \star$. The rule can actually be replaced by the following, less restrictive rule:

$$\frac{\Gamma \vdash_l P \qquad \Gamma \vdash_l Q \qquad l' \sqsubseteq l}{\Gamma \mid v:\mathbf{bool}^{l'} \vdash_l \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q}$$

This rule avoids propagation of the secrecy level of the boolean to the if-expression. If we extend the subtyping relation based on the secrecy level (as discussed in Section 6), the two rules become equivalent.

T-WEAK: This rule allows the type environment to be replaced by a type environment expressing more liberal uses of channels, and the secrecy level of the process to be replaced by a lower one. For example, from $x:\xi/O^0_\infty \vdash_{\mathbf{H}} P$, we can obtain $x:\xi/O^\infty_0 \vdash_{\mathbf{L}} P$.

3.6 Examples

Example 12 The process $(\nu x:\langle\rangle^{\mathbf{H}})\,(x().\,\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)$ is typed as follows:

$$\dfrac{\dfrac{\dfrac{x:\langle\rangle^{\mathbf{H}}/\mathbf{0}, y:\langle\rangle^{\mathbf{L}}/O^{\infty}_{\infty} \vdash_{\mathbf{L}} \overline{y}\langle\rangle}{x:\langle\rangle^{\mathbf{H}}/I^{0}_{0}, y:\langle\rangle^{\mathbf{L}}/O^{\infty}_{\infty} \vdash_{\mathbf{L}} x().\,\overline{y}\langle\rangle}\ \text{T-In}}{x:\langle\rangle^{\mathbf{H}}/I^{0}_{0}, y:\langle\rangle^{\mathbf{L}}/\uparrow^{(1,1)}O^{\infty}_{\infty} \vdash_{\mathbf{L}} x().\,\overline{y}\langle\rangle}\ \text{T-Weak} \qquad \dfrac{x:\langle\rangle^{\mathbf{H}}/O^{0}_{0} \vdash_{\mathbf{H}} \overline{x}\langle\rangle}{x:\langle\rangle^{\mathbf{H}}/O^{0}_{0} \vdash_{\mathbf{L}} \overline{x}\langle\rangle}\ \text{T-Weak}}{x:\langle\rangle^{\mathbf{H}}/(I^{0}_{0} \mid O^{0}_{0}), y:\langle\rangle^{\mathbf{L}}/O^{\infty}_{\infty} \vdash_{\mathbf{L}} x().\,\overline{y}\langle\rangle \mid \overline{x}\langle\rangle}\ \text{T-Par}}{y:\langle\rangle^{\mathbf{L}}/O^{\infty}_{\infty} \vdash_{\mathbf{L}} (\nu x:\langle\rangle^{\mathbf{H}})\,(x().\,\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)}\ \text{T-New}$$

The secrecy level of y can be $\mathbf{L}$, although that of x is declared as $\mathbf{H}$.

Example 13 Let us consider the following process P:

$$*s(r).\,\overline{r}\langle\rangle \mid \overline{s}\langle x\rangle \mid x().\,\overline{y}\langle\rangle$$

It is well typed under the following type environment:

$$s:\langle\langle\rangle^{\mathbf{H}}/O^{0}_{\infty}\rangle^{\mathbf{H}}/(*I^{0}_{\infty} \mid O^{\infty}_{0}), x:\langle\rangle^{\mathbf{H}}/(O^{1}_{\infty} \mid I^{\infty}_{1}), y:\langle\rangle^{\mathbf{L}}/O^{\infty}_{\infty}$$

Note that the type system can infer that the input on x succeeds, so that y is assigned level $\mathbf{L}$ although the secrecy level of x is $\mathbf{H}$.

Example 14 Suppose that x has type $\langle\rangle^{\mathbf{H}}/{*I^{\infty}_{n_x}}.O^{n_x}_{\infty}$ and y has type $\langle\rangle^{\mathbf{H}}/{*I^{\infty}_{n_y}}.O^{n_y}_{\infty}$ with $n_y < n_x$. Then, as mentioned in Example 9, the process $x().\,y().\,(\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)$ is allowed which locks x and y in this order, but the process $y().\,x().\,(\overline{x}\langle\rangle \mid \overline{y}\langle\rangle)$ is not. In fact, $x().\,y().\,(\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)$ is typed as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{x:\langle\rangle^{\mathbf{H}}/O^{n_x}_{\infty}, y:\langle\rangle^{\mathbf{H}}/O^{n_y}_{\infty} \vdash_{\mathbf{H}} \overline{y}\langle\rangle \mid \overline{x}\langle\rangle}}{x:\langle\rangle^{\mathbf{H}}/\uparrow^{(n_y+1,n_y+1)}O^{n_x}_{\infty}, y:\langle\rangle^{\mathbf{H}}/I^{0}_{n_y}.O^{n_y}_{\infty} \vdash_{\mathbf{H}} y().\,(\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)}\ \text{T-In}}{x:\langle\rangle^{\mathbf{H}}/O^{n_x}_{\infty}, y:\langle\rangle^{\mathbf{H}}/I^{\infty}_{n_y}.O^{n_y}_{\infty} \vdash_{\mathbf{H}} y().\,(\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)}\ \text{T-Weak}}{x:\langle\rangle^{\mathbf{H}}/I^{0}_{n_x}.O^{n_x}_{\infty}, y:\langle\rangle^{\mathbf{H}}/\uparrow^{(n_x+1,n_x+1)}I^{\infty}_{n_y}.O^{n_y}_{\infty} \vdash_{\mathbf{H}} x().\,y().\,(\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)}\ \text{T-In}}{x:\langle\rangle^{\mathbf{H}}/{*I^{\infty}_{n_x}}.O^{n_x}_{\infty}, y:\langle\rangle^{\mathbf{H}}/{*I^{\infty}_{n_y}}.O^{n_y}_{\infty} \vdash_{\mathbf{H}} x().\,y().\,(\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)}\ \text{T-Weak}$$

Here, we use the fact $U \leq \uparrow^{(t_I,t_O)}U$ when $t_O \leq ob_O(U)$ and $t_I \leq ob_I(U)$ in the applications of T-Weak.

On the other hand,

$$x:\langle\rangle^{\mathbf{H}}/{*I^{\infty}_{n_x}}.O^{n_x}_{\infty}, y:\langle\rangle^{\mathbf{H}}/{*I^{\infty}_{n_y}}.O^{n_y}_{\infty} \vdash_{\mathbf{H}} y().\,x().\,(\overline{x}\langle\rangle \mid \overline{y}\langle\rangle)$$

is not derivable. By applying T-In to $x:\langle\rangle^{\mathbf{H}}/O^{n_x}_{\infty}, y:\langle\rangle^{\mathbf{H}}/O^{n_y}_{\infty} \vdash_{\mathbf{H}} \overline{y}\langle\rangle \mid \overline{x}\langle\rangle$, we obtain

$$x:\langle\rangle^{\mathbf{H}}/I^{0}_{n_x}.O^{n_x}_{\infty}, y:\langle\rangle^{\mathbf{H}}/\uparrow^{(n_x+1,n_x+1)}O^{n_y}_{\infty} \vdash_{\mathbf{H}} x().\,(\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)$$

Since $n_y < n_x$, we cannot obtain

$$x:\langle\rangle^{\mathbf{H}}/I^{0}_{n_x}.O^{n_x}_{\infty}, y:\langle\rangle^{\mathbf{H}}/O^{n_y}_{\infty} \vdash_{\mathbf{H}} x().\,(\overline{y}\langle\rangle \mid \overline{x}\langle\rangle)$$

from the above judgment.

Example 15 The process A in Example 4 has the secrecy level $\mathbf{L}$ under the following type environment:

$secret : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / *I_0^\infty O_\infty^0, public : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}} / *I_\infty^\infty . O_\infty^\infty,$
$x : \langle\,\rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0$

(The bound channel c is assigned a type $\langle\,\rangle^{\mathbf{H}} / (O_1^1 \mid I_1^1)$.)

The process $C = y(\,).\, x(\,).\, (\overline{x}\langle\,\rangle \mid \overline{y}\langle\,\rangle)$ is well typed under:

$x : \langle\,\rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0, y : \langle\,\rangle^{\mathbf{L}} / *I_\infty^\infty . O_\infty^\infty.$

So, the process $(\nu x)\,(\nu secret)\,(A \mid C \mid \overline{x}\langle\,\rangle \mid \overline{secret}\langle b\rangle)$ is well typed where b is *true*$^{\mathbf{H}}$ or *false*$^{\mathbf{H}}$. (Here, $\overline{x}\langle\,\rangle \mid \overline{secret}\langle b\rangle$ initializes the lock x and the shared variable $secret$.) So, we know that the concurrent execution of threads A and C in Figure 1 is safe.

On the other hand, the process $B = x(\,).\, y(\,).\, (\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$ is well-typed not under the type environment $x : \langle\,\rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0, y : \langle\,\rangle^{\mathbf{L}} / *I_\infty^\infty . O_\infty^\infty$ but under $x : \langle\,\rangle^{\mathbf{H}} / *I_\infty^\infty . O_\infty^\infty, y : \langle\,\rangle^{\mathbf{L}} / *I_\infty^\infty . O_\infty^\infty$. So, $(\nu x)\,(\nu secret)\,(A \mid B \mid \overline{x}\langle\,\rangle \mid \overline{secret}\langle b\rangle)$ is not well typed (since the whole usage of x: $*I_\infty^\infty . O_\infty^\infty \mid *I_0^\infty . O_\infty^0 \mid O_\infty^0$ is not reliable), which implies that the concurrent execution of threads A and B may leak secret information.

Example 16 Let us consider the process given at the end of Example 6. Suppose that Γ_1, Γ_2 and Γ_3 are given as follows.

$\Gamma_1 = x : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0, c : \langle\,\rangle^{\mathbf{H}} / O_0^1, c_1 : \langle\,\rangle^{\mathbf{H}} / O_\infty^1$
$\Gamma_2 = x : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0, y : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0,$
$\quad c : \langle\,\rangle^{\mathbf{H}} / I_1^0, c_2 : \langle\,\rangle^{\mathbf{H}} / O_\infty^2$
$\Gamma_3 = c_1 : \langle\,\rangle^{\mathbf{H}} / I_1^0, c_2 : \langle\,\rangle^{\mathbf{H}} / I_2^2, w : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}} / *I_\infty^\infty . O_\infty^\infty$

Then, we have:

$\Gamma_1 \vdash_{\mathbf{H}} x(z).\, (\overline{x}\langle \mathit{true}^{\mathbf{H}} \rangle \mid \overline{c}\langle\,\rangle.\, \overline{c_1}\langle\,\rangle)$
$\Gamma_2 \vdash_{\mathbf{H}} c(y).\, x(z).\, (\overline{x}\langle z\rangle \mid y(u).\, (\overline{y}\langle z\rangle \mid \overline{c_2}\langle\,\rangle))$
$\Gamma_3 \vdash_{\mathbf{L}} c_1(\,).\, c_2(\,).\, w(z)\,.\, \overline{w}\langle \mathit{false}^{\mathbf{L}} \rangle$

The whole process is well-typed under the type environment:

$x : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0, y : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0, c : \langle\,\rangle^{\mathbf{H}} / (O_0^1 \mid I_1^0),$
$w : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}} / *I_\infty^\infty . O_\infty^\infty$

So, although the part `cobegin ... coend` performs synchronization on high-level channels, our type system can correctly infer that it does not affect the execution of the part `w := false`.

Example 17 Let us reconsider the process in Example 7. Subprocesses $*\mathit{withdraw}(\mathit{amount}, r).\, \cdots$, $*\mathit{getBalance}(r).\, \cdots$, and $*\mathit{deposit}(\mathit{amount}, r).\, \cdots$ are typed as follows. (Here, we assume that the type system is extended with integer types.)

$\mathit{withdraw} : \langle int^{\mathbf{H}}, \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O_\infty^1 \rangle^{\mathbf{H}} / *I_\infty^0, s : \langle int^{\mathbf{H}} \rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0$
$\qquad \vdash_{\mathbf{H}} *\mathit{withdraw}(\mathit{amount}, r).\, \cdots$
$\mathit{getBalance} : \langle \langle int^{\mathbf{H}} \rangle^{\mathbf{H}} / O_\infty^1 \rangle^{\mathbf{H}} / *I_\infty^0, s : \langle int^{\mathbf{H}} \rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0$
$\qquad \vdash_{\mathbf{H}} *\mathit{getBalance}(r).\, \cdots$
$\mathit{deposit} : \langle int^{\mathbf{L}}, \langle\,\rangle^{\mathbf{L}} / O_\infty^1 \rangle^{\mathbf{L}} / *I_\infty^0, s : \langle int^{\mathbf{H}} \rangle^{\mathbf{H}} / *I_0^\infty . O_\infty^0$
$\qquad \vdash_{\mathbf{L}} *\mathit{deposit}(\mathit{amount}, r).\, \cdots$

The key is the typing for the last sub-process. Although it performs communication on a high-level channel s, it is allowed to send a message on a low-level channel r, since our type system guarantees that the input on s always succeeds. (Note that the capability level of an input on s is 0.) The bank object is well-typed under:

$$\begin{array}{l}\mathit{withdraw} : \langle int^{\mathbf{H}}, \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^1_\infty \rangle^{\mathbf{H}} / {*} I^0_\infty,\\ \mathit{getBalance} : \langle \langle int^{\mathbf{H}} \rangle^{\mathbf{H}} / O^1_\infty \rangle^{\mathbf{H}} / {*} I^0_\infty,\\ \mathit{deposit} : \langle int^{\mathbf{L}}, \langle\,\rangle^{\mathbf{L}} / O^1_\infty \rangle^{\mathbf{L}} / {*} I^0_\infty\end{array}$$

This implies that information about the current balance is not leaked through the public channel *deposit*.

Let us now consider the client process given at the end of Example 7. The part $\overline{m}\langle 10, r\rangle . r(b) . \mathbf{0}$ is typed under:

$$\begin{array}{l} m : \langle int^{\mathbf{H}}, \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^1_\infty \rangle^{\mathbf{H}} / {*} O^\infty_0,\\ r : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / (O^1_\infty \mid I^\infty_1).\end{array}$$

So, the whole client process is typed under:

$$\begin{array}{l}\mathit{withdraw} : \langle int^{\mathbf{H}}, \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^1_\infty \rangle^{\mathbf{H}} / {*} O^\infty_0,\\ y : \langle \langle int^{\mathbf{H}}, \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^1_\infty \rangle^{\mathbf{H}} / {*} O^\infty_0 \rangle^{\mathbf{H}} / {*} I^\infty_0 . O^0_\infty.\end{array}$$

Example 18 The following process implements the parallel-or:

$$\begin{array}{l} P \stackrel{\triangle}{=} {*}\mathit{por}(f_1, f_2, r).\\ \quad (\nu x)\,(\overline{f_1}\langle x\rangle \mid \overline{f_2}\langle x\rangle \mid x(b_1).\,\mathbf{if}\ b_1\ \mathbf{then}\ \overline{r}\langle \mathit{true}\rangle\ \mathbf{else}\ x(b_2).\,\overline{r}\langle b_2\rangle)\end{array}$$

It receives a triple $[f_1, f_2, r]$, where f_1 and f_2 are the locations of processes that return booleans, and r is a channel that should be used for returning the result. Upon receiving $[f_1, f_2, r]$, P creates a new channel x and sends them to f_1 and f_2. It then waits for a reply on channel x. Two booleans are expected to arrive on x, but if the value received first is *true*, the process returns *true* without waiting for the second value.

For example, consider the following process Q. Q sends on por the locations t and f of servers that always answer *true* and *false* respectively, and then waits for the result.

$$Q \stackrel{\triangle}{=} {*}t(r).\,\overline{r}\langle \mathit{true}\rangle \mid {*}f(r).\,\overline{r}\langle \mathit{false}\rangle \mid (\nu y)\,(\overline{\mathit{por}}\langle t, f, y\rangle \mid y(b).\,\overline{\mathit{succ}}\langle\,\rangle)$$

$P \mid Q$ is well-typed under the following type environment:

$$\begin{array}{l} t, f : \langle \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^0_\infty \rangle^{\mathbf{H}} / ({*} I^0_\infty \mid {*} O^\infty_0),\\ \mathit{por} : \langle \tau, \tau, \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^2_\infty \rangle^{\mathbf{H}} / ({*} I^0_\infty \mid {*} O^\infty_0),\\ \mathit{succ} : \langle\,\rangle^{\mathbf{L}} / O^4_\infty\\ \text{where } \tau \stackrel{\triangle}{=} \langle \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^0_\infty \rangle^{\mathbf{H}} / O^\infty_0.\end{array}$$

The bound variable x in P is given the following type:

$$\langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / (O^1_\infty \mid O^1_\infty \mid I^0_1 . (\mathbf{0}\ \&\ I^0_1)).$$

The usage indicates that while outputs on x may not succeed (the capability level is ∞), the inputs on x always succeed.

4 Soundness of the Type System

In this section, we show that well-typed processes satisfy a so-called non-interference property, which says that high-level values and processes (i.e., values and processes of the secrecy level **H**) do not affect the behavior of low-level processes. The property implies that information about high-level values or processes cannot be observed by low-level processes.

Before proving the non-interference property, we show a subject reduction theorem. As in other linear type systems for process calculi [17], type environments may change during reduction. We write $\Gamma \longrightarrow \Gamma'$ when $\Gamma = \Gamma_1, x:\xi/U$ and $\Gamma' = \Gamma_1, x:\xi/U'$ with $U \longrightarrow U'$ for some Γ_1, x, ξ, U, and U'.

Theorem 1 (subject reduction) *If $\Gamma \vdash_l P$ and $P \longrightarrow Q$, then $\Delta \vdash_l Q$ holds for some Δ such that $\Gamma = \Delta$ or $\Gamma \longrightarrow \Delta$.*

Proof See Appendix A.

In order to formally state the non-interference property, we define a process equivalence relation based on the notion of barbed congruence. The idea of barbed congruence is to put two processes into various contexts and check whether they exhibit the same observational behavior. The set of observables, called *barbs*, is defined as follows.

Definition 21 (barbs) The *barbs* of P, written $\mathit{Barbs}(P)$, is defined by:

$$\begin{aligned}\mathit{Barbs}(P) = \{&\overline{x} \mid P \longrightarrow^* \preceq (\nu\widetilde{y})\,(\overline{x}\langle\widetilde{v}\rangle.\,Q \mid R), x \notin \{\widetilde{y}\}\}\\ \cup\{&x \mid P \longrightarrow^* \preceq (\nu\widetilde{y})\,(x(\widetilde{z}).\,Q \mid R), x \notin \{\widetilde{y}\}\}\end{aligned}$$

The two processes put into the same context are compared by using the following barbed bisimulation.

Definition 22 (barbed bisimulation) A binary relation $\mathcal{R}$ on processes is a *barbed bisimulation* if the following conditions hold for every $(P, Q) \in \mathcal{R}$:

- If $P \longrightarrow P'$, then there exists Q' such that $Q \longrightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$,
- If $Q \longrightarrow Q'$, then there exists P' such that $P \longrightarrow^* P'$ and $(P', Q') \in \mathcal{R}$, and
- $\mathit{Barbs}(P) = \mathit{Barbs}(Q)$.

P and Q are barbed bisimilar, written $P \stackrel{\bullet}{\approx} Q$, if $(P, Q) \in \mathcal{R}$ holds for some barbed bisimulation.

The definition of contexts is given as follows.

Definition 23 (context) A *context* is a term obtained from a process by replacing a sub-process with $[\,]$. We write $C[P]$ for the process obtained by replacing $[\,]$ in C with P. A context C is a (Γ, l)-(Δ, l')-context if $\Delta \vdash_{l'} C$ is derivable from $\Gamma \vdash_l [\,]$.

We introduce some terminology about type environments. A type environment Γ is *low-level* if all the secrecy level annotations appearing in Γ are **L**. A type environment Γ is *closed* [30] if $\Gamma(x)$ is a channel type for any $x \in dom(\Gamma)$. Γ is *reliable*, written $rel(\Gamma)$, if for any $x \in dom(\Gamma)$, $\Gamma(x)$ is a channel type of the form ξ/U and $rel(U)$ holds.

Now we can define the barbed congruence. Basically, two processes P and Q are barbed congruent if $C[P]$ and $C[Q]$ are barbed bisimilar for an arbitrary context C. Here, since we are dealing with well-typed processes, we consider only "well-typed" contexts.

Definition 24 (barbed congruence) P and Q are barbed (Γ, l)-congruent, written $P \approx_{\Gamma,l} Q$, if (i) $\Gamma \vdash_l P$, (ii) $\Gamma \vdash_l Q$, and (iii) for any closed Δ and secrecy level l', $C[P] \stackrel{\bullet}{\approx} C[Q]$ holds for any (Γ, l)-(Δ, l')-context C.

We can now state the non-interference property as the following theorems. The first one says that the difference between high-level *values* is not observable to low-level processes, and the second one says that the difference between high-level *processes* is not observable to low-level processes. The second one is required to prevent leakage of information about complex data structures, which are represented as processes in the π-calculus [30].

Theorem 2 (non-interference (1)) *Suppose that Γ is a low-level type environment. If $\Gamma \vdash_l [x \mapsto \mathit{true}^{\mathbf{H}}]P$ holds, then $[x \mapsto \mathit{true}^{\mathbf{H}}]P \approx_{\Gamma,l} [x \mapsto \mathit{false}^{\mathbf{H}}]P$.*

Theorem 3 (non-interference (2)) *Suppose that Γ is a low-level type environment, and that C_1 is a $(\Theta, \mathbf{H})$-(Γ, l)-context. If $\Theta \vdash_{\mathbf{H}} P_i$ holds for $i = 1, 2$, then $C_1[P_1] \approx_{\Gamma,l} C_1[P_2]$.*

The rest of this section is devoted to proofs of Theorems 2 and 3. The central idea of the proofs is that if $\Delta \vdash_l P$ holds (with a certain condition on Δ), then P and the process $\mathbf{Er}(P)$ obtained from P by eliminating all the high-level values and processes are barbed bisimilar (Theorem 4). In the case of Theorem 2, for any $(\Gamma, l) - (\Delta, l')$ context C, $C[[x \mapsto \mathit{true}^{\mathbf{H}}]P]$ and $C[[x \mapsto \mathit{false}^{\mathbf{H}}]P]$ are erased to the same process (up to a structural relation), so that $C[[x \mapsto \mathit{true}^{\mathbf{H}}]P] \stackrel{\bullet}{\approx} \mathbf{Er}(C[[x \mapsto \mathit{true}^{\mathbf{H}}]P]) \stackrel{\bullet}{\approx} \mathbf{Er}(C[[x \mapsto \mathit{false}^{\mathbf{H}}]P]) \stackrel{\bullet}{\approx} C[[x \mapsto \mathit{false}^{\mathbf{H}}]P]$ holds.

Below we define $\mathbf{Er}$ formally and then prove the non-interference theorems.

We write $\mathbf{High}(\tau)$ if τ is $\mathbf{unit}$, $\mathbf{bool}^{\mathbf{H}}$, or a channel type of the form $\langle \widetilde{\tau} \rangle^{\mathbf{H}}/U$.

Definition 25 $\mathbf{Er}_\Gamma(P)$ is defined by:

$$\mathbf{ErV}_\Gamma(v) = \begin{cases} \star \text{ if } \mathbf{High}(\Gamma(v)) \\ v \text{ otherwise} \end{cases}$$

$$\mathbf{Er}_\Gamma(\mathbf{0}) = \mathbf{0}$$

$$\mathbf{Er}_\Gamma(\overline{x}\langle \widetilde{v} \rangle.\, P) = \begin{cases} \overline{x}\langle \widetilde{v'} \rangle.\, \mathbf{Er}_\Gamma(P) \\ \qquad \text{if } \Gamma(x) = \langle \widetilde{\tau} \rangle^{\mathbf{L}}/U \text{ and } v'_i = \mathbf{ErV}_\Gamma(v_i) \\ \mathbf{Er}_\Gamma(P) \qquad \text{otherwise} \end{cases}$$

$$\mathbf{Er}_\Gamma(x(\widetilde{y}).\, P) = \begin{cases} x(\widetilde{y}).\, \mathbf{Er}_{\Gamma, \widetilde{y}:\widetilde{\tau}}(P) & \text{if } \Gamma(x) = \langle \widetilde{\tau} \rangle^{\mathbf{L}}/U \\ \mathbf{Er}_{\Gamma, \widetilde{y}:\widetilde{\tau}}(P) & \text{if } \Gamma(x) = \langle \widetilde{\tau} \rangle^{\mathbf{H}}/U \\ \mathbf{Er}_{\Gamma, \widetilde{y}:\widetilde{\mathbf{unit}}}(P) & \text{otherwise} \end{cases}$$

$$\mathbf{Er}_\Gamma(P \mid Q) = \mathbf{Er}_\Gamma(P) \mid \mathbf{Er}_\Gamma(Q)$$

$$\mathbf{Er}_\Gamma(*P) = *\mathbf{Er}_\Gamma(P)$$

$$\mathbf{Er}_\Gamma((\nu x : \xi)\, P) = \begin{cases} (\nu x : \xi)\, \mathbf{Er}_{\Gamma, x:\xi/\mathbf{0}}(P) & \text{if } \xi = \langle \widetilde{\tau} \rangle^{\mathbf{L}} \\ \mathbf{Er}_{\Gamma, x:\xi/\mathbf{0}}(P) & \text{otherwise} \end{cases}$$

$$\mathbf{Er}_\Gamma(\mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q) = \begin{cases} \mathbf{if}\ v\ \mathbf{then}\ \mathbf{Er}_\Gamma(P)\ \mathbf{else}\ \mathbf{Er}_\Gamma(Q) \\ \qquad \text{if } \Gamma(v) = \mathbf{bool}^{\mathbf{L}} \\ \mathbf{0} \quad \text{otherwise} \end{cases}$$

The following lemma shows that all high-level processes are erased to $\mathbf{0}$. (Recall that $P \equiv Q$ means $P \preceq Q$ and $Q \preceq P$.)

Lemma 4 *If $\Gamma \vdash_{\mathbf{H}} P$, then $\mathbf{Er}_\Gamma(P) \equiv \mathbf{0}$.*

Proof Straightforward induction on derivation of $\Gamma \vdash_{\mathbf{H}} P$.

The following is a key theorem, which states that the erasure function preserves barbed bisimilarity.

Theorem 4 *Suppose that Γ is a low-level, reliable type environment. If $\Gamma \vdash_l P$, then $P \overset{\bullet}{\approx} \mathbf{Er}_\Gamma(P)$.*

The proof the above theorem is given in Appendix B. Here, we explain informally why the above theorem holds. The difference between P and $\mathbf{Er}_\Gamma(P)$ is that all the input and output prefixes on high-level channels of P are removed in $\mathbf{Er}_\Gamma(P)$. The behavior of P can be simulated by $\mathbf{Er}_\Gamma(P)$ can be simulated as follows.

- Any reduction of P on a low-level channel is matched by the corresponding reduction of $\mathbf{Er}_\Gamma(P)$.
- Any reduction of P on a high-level channel is matched by the skip $\mathbf{Er}_\Gamma(P) \longrightarrow^* \mathbf{Er}_\Gamma(P)$.

For example, let us consider a process $P = (\nu x : \langle\,\rangle^{\mathbf{H}})\,(x(\,).\,\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle.\,y(\,))$ and a type environment $\Gamma = y : \langle\,\rangle^{\mathbf{L}}/(O_\infty^\infty \mid I_\infty^\infty)$. $\mathbf{Er}_\Gamma(P) = \overline{y}\langle\,\rangle \mid y(\,)$. The reduction $P \longrightarrow (\nu x : \langle\,\rangle^{\mathbf{H}})\,(\overline{y}\langle\,\rangle \mid y(\,))$ is matched by the skip $\overline{y}\langle\,\rangle \mid y(\,) \longrightarrow^* \overline{y}\langle\,\rangle \mid y(\,)$.

The simulation of $\mathbf{Er}_\Gamma(P)$ by P needs more attention. Since some prefixes of P have been removed in $\mathbf{Er}_\Gamma(P)$, some communications enabled in $\mathbf{Er}_\Gamma(P)$ may not be enabled in P. In the example above, the communication on y is enabled in $\mathbf{Er}_\Gamma(P)$, but not in P. However, because of the typing rules T-OUT and T-IN (the condition $t_c = \infty \Rightarrow l_1 \sqsubseteq l_2$), all the input/output prefixes that are blocking communications that are enabled in $\mathbf{Er}_\Gamma(P)$ but not in P have capabilities of finite level (that is, t_c is finite in T-OUT and T-IN). Since those input/output operations eventually succeed (see Theorem 5 in Appendix B), the communications that are enabled in $\mathbf{Er}_\Gamma(P)$ are also eventually enabled in P. For example, in the case of the above example, the reduction $\mathbf{Er}_\Gamma(P) = \overline{y}\langle\,\rangle \mid y(\,) \longrightarrow \mathbf{0}$ can be simulated by $P = (\nu x : \langle\,\rangle^{\mathbf{H}})\,(x(\,).\,\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle.\,y(\,)) \longrightarrow (\nu x : \langle\,\rangle^{\mathbf{H}})\,(\overline{y}\langle\,\rangle \mid y(\,)) \longrightarrow (\nu x : \langle\,\rangle^{\mathbf{H}})\,\mathbf{0}$. See Appendix B for a more detailed proof.

We now move on to prepare for the proofs of the non-interference theorems. We write $\mathbf{low}(\Gamma)$ for the type environment obtained from Γ by replacing all the secrecy annotations with $\mathbf{L}$ and all the capability level annotations with ∞. We also write $\mathbf{low}(C)$ for the context obtained from C by replacing all the secrecy annotations with $\mathbf{L}$ and all the capability level annotations with ∞. For example, $\mathbf{low}((\nu x : \langle\langle\,\rangle^{\mathbf{H}}/O_1^0\rangle^{\mathbf{L}})\,(\mathbf{if}\ \mathit{true}^{\mathbf{H}}\ \mathbf{then}\ \overline{x}\langle y\rangle\ \mathbf{else}\ [\,])) = (\nu x : \langle\langle\,\rangle^{\mathbf{L}}/O_\infty^0\rangle^{\mathbf{L}})\,(\mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ \overline{x}\langle y\rangle\ \mathbf{else}\ [\,])$.

Lemma 5 *Suppose that Γ is a low-level type environment. If C is a (Γ, l)-(Δ, l')-context, then $\mathbf{low}(C)$ is a (Γ, l)-$(\mathbf{low}(\Delta), \mathbf{L})$-context.*

Proof This follows by straightforward induction on the derivation of $\Delta \vdash_{l'} C$.

The erasure function is extended to that on contexts by $\mathbf{Er}_\Gamma([\,]) = [\,]$. The following lemma follows by straightforward induction on the structure of C.

Lemma 6 *Suppose that $\Gamma \vdash_l P$ and that C is a $(\Gamma, l) - (\Delta, l')$ context. Then, $\mathbf{Er}_\Delta(C[P]) = \mathbf{Er}_\Delta(C)[\mathbf{Er}_\Gamma(P)]$.*

Now we can prove the non-interference theorems.

Proof of Theorem 2 Suppose that Δ is closed. Then $\mathbf{low}(\Delta)$ is a low-level, reliable type environment (since all the capability level annotations have been replaced by ∞). Let C be a (Γ, l)-(Δ, l')-context, and C' be $\mathbf{low}(C)$. By Lemma 5, C' is a (Γ, l)-$(\mathbf{low}(\Delta), \mathbf{L})$-context. By Lemma 4 and Theorem 4, $C'[[x \mapsto \mathit{true}^{\mathbf{H}}]P] \stackrel{\bullet}{\approx} \mathbf{Er}_{\mathbf{low}(\Delta)}(C'[[x \mapsto \mathit{true}^{\mathbf{H}}]P]) = \mathbf{Er}_{\mathbf{low}(\Delta)}(C')[\mathbf{Er}_\Gamma([x \mapsto \mathit{true}^{\mathbf{H}}]P)] = \mathbf{Er}_{\mathbf{low}(\Delta)}(C')[\mathbf{Er}_\Gamma([x \mapsto \mathit{false}^{\mathbf{H}}]P)] = \mathbf{Er}_{\mathbf{low}(\Delta)}(C'[[x \mapsto \mathit{false}^{\mathbf{H}}]P]) \stackrel{\bullet}{\approx} C'[[x \mapsto \mathit{false}^{\mathbf{H}}]P]$. Since $\stackrel{\bullet}{\approx}$ does not depend on type annotation, $C[[x \mapsto \mathit{true}^{\mathbf{H}}]P] \stackrel{\bullet}{\approx} C[[x \mapsto \mathit{false}^{\mathbf{H}}]P]$ holds. □

Proof of Theorem 3 Suppose that Δ is closed. Then $\mathbf{low}(\Delta)$ is a low-level, reliable type environment. Let C be a (Γ, l)-(Δ, l')-context, and C' be $\mathbf{low}(C)$. By Lemma 5, C' is a (Γ, l)-$(\mathbf{low}(\Delta), \mathbf{L})$-context. By Lemma 4 and Theorem 4,

$$\begin{aligned}
C'[C_1[P_1]] &\stackrel{\bullet}{\approx} \mathbf{Er}_{\mathbf{low}(\Delta')}(C'[C_1[P_1]]) \\
&= \mathbf{Er}_{\mathbf{low}(\Delta)}(C'[C_1])[\mathbf{Er}_\Theta(P_1)] \\
&\stackrel{\bullet}{\approx} \mathbf{Er}_{\mathbf{low}(\Delta)}(C'[C_1])[\mathbf{0}] \\
&\stackrel{\bullet}{\approx} \mathbf{Er}_{\mathbf{low}(\Delta)}(C'[C_1])[\mathbf{Er}_\Theta(P_2)] \\
&= \mathbf{Er}_{\mathbf{low}(\Delta)}(C'[C_1[P_2]]) \\
&\stackrel{\bullet}{\approx} C'[C_1[P_2]].
\end{aligned}$$

Since $\stackrel{\bullet}{\approx}$ does not depend on type annotation, $C[C_1[P_1]] \stackrel{\bullet}{\approx} C[C_1[P_2]]$ holds. □

5 Type Inference Algorithm

This section describes a type inference algorithm. By the soundness of the type system, the type inference algorithm can be used to check that processes do not leak secret information. An input of the algorithm is a triple (Γ, P, l), where types and secrecy levels appearing in the triple may contain variables (to represent unknown types, secrecy levels, etc.), and all the usages in the triple must be variables. The algorithm answers whether there exists a substitution θ such that $\theta\Gamma \vdash_{\theta l} \theta P$. (The algorithm can be modified to also output a set of constraints that contain all the correct substitutions.) The algorithm is sound and complete with respect to the type system, in the sense that it always terminates and gives a correct yes/no-answer.

Thanks to the type inference algorithm, programmers only need to put annotations only when they want to explicitly specify values that should be regarded as secret or assumptions about the behavior of the environment (by specifying a part of information about Γ). Since the input Γ, P, l of the type inference algorithm may contain variables, programmers are *not* obliged to provide any annotation. In the extreme case, a programmer can write an ordinary pi-calculus term (without annotations on ν-prefixes and truth values in Definition 2). Then, a system can

automatically insert variables in the places where annotations are required according to the syntax of Definition 2, and use the annotated process as an input to the inference algorithm, along with a dummy type environment $x_1 : \alpha_1, \ldots, x_n : \alpha_n$, where $x_1, \ldots, x_n$ are the free variables of the process and $\alpha_1, \ldots, \alpha_n$ are unknown type variables. Then, the inference algorithm tries to assign $\mathbf{H}$ to as many values as possible.

The overall structure of the type inference algorithm is similar to other constraint-based type inference algorithms for the π-calculus [12, 13, 18]: Given an input (Γ, P, l), we can first obtain a set $\mathcal{C}$ of constraints on type variables, usage variables, etc. such that $\theta\mathcal{C}$ holds if and only if $\theta\Gamma \vdash_{\theta l} \theta P$ holds. Then, we can reduce $\mathcal{C}$ step by step to decide whether $\mathcal{C}$ is satisfiable.

In the rest of this section, we first show an algorithm to extract constraints in Section 5.1, and then explain how to solve the constraints in Section 5.2. Section 5.3 gives some examples to illustrate how the type inference algorithm works, and Section 5.4 briefly explains our prototype implementation of the type inference algorithm.

5.1 Step 1: Extracting constraints based on syntax-directed rules

We can convert the typing rules to syntax-directed ones, by combining each typing rule with T-WEAK. For example, the rule for parallel composition becomes:

$$\frac{\Gamma_1 \vdash^{\mathbf{SD}}_{l_1} P_1 \qquad \Gamma_2 \vdash^{\mathbf{SD}}_{l_2} P_2 \qquad l \sqsubseteq l_1, l_2}{\Gamma_1 \mid \Gamma_2 \vdash^{\mathbf{SD}}_{l} P_1 \mid P_2}$$

The whole set of syntax-directed rules is given in Figure 3. It is equivalent to the original typing rules in the following sense.

Lemma 7 *If $\Gamma \vdash_l P$, then there exist Γ' and l' such that $\Gamma' \vdash^{\mathbf{SD}}_{l'} P$ with $\Gamma \leq \Gamma'$ and $l \sqsubseteq l'$. If $\Gamma \vdash^{\mathbf{SD}}_{l} P$, then $\Gamma \vdash_l P$.*

Proof Straightforward induction on derivations of $\Gamma \vdash_l P$ and $\Gamma \vdash^{\mathbf{SD}}_{l} P$. □

An algorithm $Tinf$ for extracting constraints is shown in Figure 4. In the figure, $\mathcal{C}^{\leq}(\Gamma_1, \Gamma_2)$ denotes the set $\{\Gamma_1(x) \leq \Gamma_2(x) \mid x \in dom(\Gamma_2)\} \cup \{noob(\Gamma_1(x)) \mid x \notin dom(\Gamma_2)\}$ of constraints on types. The constraint $noob(\tau)$ in ST-IN denotes the same constraint as $ob(\tau) = \infty$, but $noob(\langle\widetilde{\tau}\rangle^l/U)$ should be reduced to $U \leq \mathbf{0}$ rather than $ob(U) = \infty$ (since we want to generate only restricted forms of constraints to simplify the reduction of constraints in the next step). The constraint $WF(\Gamma)$ means that $\Gamma(x)$ is well-formed for every $x \in dom(\Gamma)$. $WF(\tau)$ means that τ is well-formed. Meta-variables β, δ, and η represent variables denoting unknown types, secrecy levels, and capability/obligation levels respectively.

The sub-algorithm PT takes P as an input, and outputs a triple $(\Gamma', l', \mathcal{C}')$ that satisfies: (i) $\theta\Gamma' \vdash_{\theta l'} \theta P$ holds for any substitution θ such that $\theta\mathcal{C}'$ holds, and (ii) if $\Gamma'' \vdash_{l''} \theta'' P$, then there exists a substitution θ such that $\theta\mathcal{C}'$, $\Gamma'' \leq \theta\Gamma'$, $l'' \sqsubseteq \theta l'$, and $\theta'' P = \theta P$. Since $PT(P)$ collects the premises of each syntax-directed rule, the following corollary immediately follows from Lemma 7.

Corollary 1 *Let $Tinf(\Gamma, l, P) = \mathcal{C}$. Then, $\theta\mathcal{C}$ holds if and only if $\theta\Gamma \vdash_{\theta l} \theta P$ holds.*

$$\emptyset \vdash_l^{\mathbf{SD}} \mathbf{0} \qquad \text{(ST-ZERO)}$$

$$\frac{\Gamma \vdash_{l_2}^{\mathbf{SD}} P \qquad l \sqsubseteq l_1, l_2 \qquad \Gamma(x) = \langle\widetilde{\tau}\rangle^{l_1}/U \text{ or } (x \notin dom(\Gamma) \text{ and } U \leq \mathbf{0}) \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\uparrow^{(t_c+1,t_c+1)}(\Gamma\backslash\{x\} \mid \widetilde{v} : \uparrow\widetilde{\tau}) \mid x : \langle\widetilde{\tau}\rangle^{l_1}/O_{t_c}^0.U \vdash_l^{\mathbf{SD}} \overline{x}\langle\widetilde{v}\rangle.\, P} \qquad \text{(ST-OUT)}$$

$$\frac{\Gamma \vdash_{l_2}^{\mathbf{SD}} P \qquad l \sqsubseteq l_1, l_2 \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2 \qquad \Gamma(x) = \langle\widetilde{\tau}\rangle^{l_1}/U \text{ or } (x \notin dom(\Gamma) \text{ and } U \leq \mathbf{0}) \qquad \tau_i \leq \Gamma(y_i) \text{ or } (y_i \notin dom(\Gamma) \text{ and } ob(\tau_i) = \infty) \text{ (for each } \tau_i \in \{\widetilde{\tau}\})}{\uparrow^{(t_c+1,t_c+1)}\Gamma\backslash\{x\}, x : \langle\widetilde{\tau}\rangle^{l_1}/I_{t_c}^0.U \vdash_l^{\mathbf{SD}} x(\widetilde{y}).\, P} \qquad \text{(ST-IN)}$$

$$\frac{\Gamma_1 \vdash_{l_1}^{\mathbf{SD}} P_1 \qquad \Gamma_2 \vdash_{l_2}^{\mathbf{SD}} P_2 \qquad l \sqsubseteq l_1, l_2}{\Gamma_1 \mid \Gamma_2 \vdash_l^{\mathbf{SD}} P_1 \mid P_2} \qquad \text{(ST-PAR)}$$

$$\frac{\Gamma \vdash_l^{\mathbf{SD}} P \qquad (\Gamma(x) = \xi/U \wedge rel(U)) \text{ if } x \in dom(\Gamma)}{\Gamma\backslash x \vdash_l^{\mathbf{SD}} (\nu x : \xi)\, P} \qquad \text{(ST-NEW)}$$

$$\frac{\Gamma_i \vdash_{l_i}^{\mathbf{SD}} P_i \text{ (for } i = 1, 2) \qquad \Gamma \leq \Gamma_i \qquad l \sqsubseteq l_i \text{ (for } i = 1, 2)}{\Gamma \mid v : \mathbf{bool}^l \vdash_l^{\mathbf{SD}} \mathbf{if}\ v\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2} \qquad \text{(ST-IF)}$$

$$\frac{\Gamma \vdash_l^{\mathbf{SD}} P}{*\Gamma \vdash_l^{\mathbf{SD}} *P} \qquad \text{(ST-REP)}$$

Fig. 3 Syntax-Directed Typing Rules

5.2 Reducing constraints

By reducing the constraints on types generated by the algorithm in the previous step, we obtain constraints on usages and secrecy/capability/obligation levels of the following forms:

$$\rho \leq U \qquad l_1 \sqsubseteq l_2 \qquad rel(U)$$
$$\eta = \infty \Rightarrow l_1 \sqsubseteq l_2 \qquad l = \mathbf{L} \Rightarrow \mathbf{Cap}_\infty(U)$$

Here, $\mathbf{Cap}_\infty(U)$ in the last constraint means that all the capability levels appearing in U are ∞. The constraint comes from the well-formedness condition on types (recall Definition 14). The meta-variable l represents $\mathbf{L}$, $\mathbf{H}$, or a variable (called *secrecy variables*) representing an unknown secrecy level. The meta-variable η represents a variable (called *level variables*) representing an unknown obligation/capability level. Usage U in constraints may contain the operation $\uparrow$ (in addition to usage constructors), and all the capability level annotations in U are level variables.

Constraints can be further reduced step by step as described below.

5.2.1 Solving subusage constraints

A set of subusage constraints $\{\rho_1 \leq U_1, \ldots, \rho_n \leq U_n\}$ is solved in the following two steps:

1. Transform the constraints so that each usage variable appears exactly once in the lefthand side of the inequalities. This is achieved by replacing $\{\rho \leq$

$Tinf(\Gamma, l, P) =$
  let $(\Gamma', l', \mathcal{C}') = PT(P)$
  in $\mathcal{C}' \cup \mathcal{C}^{\leq}(\Gamma, \Gamma') \cup \{l \sqsubseteq l'\} \cup WF(\Gamma)$

$PT(\mathbf{0}) = (\emptyset, \delta, \emptyset)$
  (where δ is fresh)

$PT(\overline{x}\langle v_1, \ldots, v_n\rangle.\, P_0) =$
  let $(\Gamma_0, l_0, \mathcal{C}_0) = PT(P_0)$
    $\mathcal{C}_1 =$ **if** $x \in dom(\Gamma_0)$ **then** $\{\Gamma_0(x) = \langle\beta_1, \ldots, \beta_n\rangle^{\delta_1}/\rho\}$ **else** $\{\rho \leq \mathbf{0}\}$
    $\mathcal{C} = \mathcal{C}_0 \cup \mathcal{C}_1 \cup \{\delta \sqsubseteq \delta_1, \delta \sqsubseteq l_0, \eta = \infty \Rightarrow \delta_1 \sqsubseteq l_0\}$
    in $(\uparrow^{(\eta+1,\eta+1)}(\Gamma_0 \backslash \{x\} \mid v_1 : \uparrow\beta_1 \mid \cdots \mid v_n : \uparrow\beta_n) \mid x : \langle\widetilde{\beta}\rangle^{\delta_1}/O^0_\eta.\rho, \delta,$
      $\mathcal{C} \cup \{WF(\langle\widetilde{\beta}\rangle^{\delta_1}/O^0_\eta.\rho)\})$
  (where $\beta_1, \ldots, \beta_n, \delta, \delta_1, \eta, \rho$ are fresh)

$PT(x(y_1, \ldots, y_n).\, P_0) =$
  let $(\Gamma_0, l_0, \mathcal{C}_0) = PT(P_0)$
    $\mathcal{C}_1 =$ **if** $x \in dom(\Gamma_0)$ **then** $\{\Gamma_0(x) = \langle\beta_1, \ldots, \beta_n\rangle^{\delta_1}/\rho\}$ **else** $\{\rho \leq \mathbf{0}\}$
    $\mathcal{C} = \mathcal{C}_0 \cup \mathcal{C}_1 \cup \{\delta \sqsubseteq \delta_1, \delta \sqsubseteq l_0, \eta = \infty \Rightarrow \delta_1 \sqsubseteq l_0\}$
      $\cup\{\beta_i \leq \Gamma_0(y_i) \mid y_i \in \{y_1, \ldots, y_n\} \cap dom(\Gamma_0)\}$
      $\cup\{noob(\beta_i) \mid y_i \in \{y_1, \ldots, y_n\} \backslash dom(\Gamma_0)\}$
  in $((\uparrow^{(\eta+1,\eta+1)}\Gamma_0 \backslash \{x, y_1, \ldots, y_n\}, x : \langle\widetilde{\beta}\rangle^{\delta_1}/I^0_\eta.\rho), \delta, \mathcal{C} \cup \{WF(\langle\widetilde{\beta}\rangle^{\delta_1}/I^0_\eta.\rho)\})$
  (where $\beta_1, \ldots, \beta_n, \delta, \delta_1, \eta, \rho$ are fresh)

$PT(P_1 \mid P_2) =$
  let $(\Gamma_1, l_1, \mathcal{C}_1) = PT(P_1)$
    $(\Gamma_2, l_2, \mathcal{C}_2) = PT(P_2)$
  in $(\Gamma_1 \mid \Gamma_2, \delta, \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{\delta \sqsubseteq l_1, \delta \sqsubseteq l_2\})$

$PT((\nu x : \xi)\, P_0) =$
  let $(\Gamma_0, l_0, \mathcal{C}_0) = PT(P_0)$
    $\mathcal{C}_1 =$ **if** $x \in dom(\Gamma_0)$ **then** $\{\Gamma_0(x) = \xi/\rho, rel(\rho)\}$ **else** $\emptyset$
  in $(\Gamma_0 \backslash \{x\}, l_0, \mathcal{C}_0 \cup \mathcal{C}_1 \cup \{WF(\xi/\mathbf{0})\})$
  (where ρ is fresh)

$PT(\mathbf{if}\ v\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2) =$
  let $(\Gamma_1, l_1, \mathcal{C}_1) = PT(P_1)$
    $(\Gamma_2, l_2, \mathcal{C}_2) = PT(P_2)$
    $\Gamma = x_1 : \beta_1, \ldots, x_n : \beta_n$
     where $\{x_1, \ldots, x_n\} = dom(\Gamma_1) \cup dom(\Gamma_2)$ and $\beta_1, \ldots, \beta_n$ are fresh
  in $(\Gamma \mid v : \mathbf{bool}^\delta, \delta, \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{\delta \sqsubseteq l_1, \delta \sqsubseteq l_2\} \cup \mathcal{C}^{\leq}(\Gamma, \Gamma_1) \cup \mathcal{C}^{\leq}(\Gamma, \Gamma_2))$
  (where δ is fresh)

$PT(*P_0) =$
  let $(\Gamma_0, l_0, \mathcal{C}_0) = PT(P_0)$
  in $(*\Gamma_0, l_0, \mathcal{C}_0)$

Fig. 4 A Type Inference Algorithm

$U_1, \rho \leq U_2\}$ with $\{\rho \leq U_1 \,\&\, U_2\}$, and add $\rho \leq \rho$ for each usage variable ρ that does not appear in the lefthand side.
2. Eliminate subusage constraints by repeatedly applying the rule: $\mathcal{C} \cup \{\rho \leq U\} \longrightarrow [\rho \mapsto \mu\rho.U]\mathcal{C}$.

The set of reduced constraints is satisfiable if and only if the original set of constraints is satisfiable, since by Lemma 3, $\rho = \mu\rho.U$ is the greatest (with respect to $\leq$) solution for $\rho \leq U$. Note that the constraints other than subusage constraints are monotonic with respect to $\leq$ in the sense that if $U_1 \leq U_2$, then $[\rho \mapsto U_1]\mathcal{C}$ implies $[\rho \mapsto U_2]\mathcal{C}$.

5.2.2 Removing the operator $\uparrow$

Usages in the set of constraints obtained so far may contain the operator $\uparrow$. $\uparrow U$ can be replaced by $\uparrow^{(ob_I(U)+1, ob_O(U)+1)}U$. We can show by induction on the structure of U that $ob_\alpha(U)$ is expressed in terms of $\mathbf{min}$, $\mathbf{max}$, constants (in $\mathbf{Nat} \cup \{\infty\}$), and expressions of the form $\eta + n$ where η is a level variable. The non-trivial is the case where U is of the form $\mu\rho.U_1$. By the definition of ob_α, $ob_\alpha(U) = \mathbf{lfp}(\lambda x.ob_\alpha^{\{\rho \mapsto x\}}(U_1))$. By induction hypothesis, $ob_\alpha^{\{\rho \mapsto x\}}(U_1)$ can be normalized to:

$$\mathbf{max}(e_1, \ldots, e_n)$$

where each e_i is of the form c_i or $\mathbf{min}(x + n_i, c_i)$. Here, $n_i \in \mathbf{Nat}$ and c_i is an expression not containing x. If $n > 0$, then

$$\mathbf{lfp}(\lambda x.\mathbf{max}(\mathbf{min}(x + n, c), e)) = \mathbf{lfp}(\lambda x.\mathbf{max}(c, e))$$

(Note that $\mathbf{max}(\mathbf{min}(x + n, c), e)$ and $\mathbf{max}(c, e)$ can differ only in the range $0 \leq x < c - n$, and that there is no fixpoint in that range.) So, we can assume without loss of generality that $ob_\alpha^{\{\rho \mapsto x\}}(U_1)$ is of the form $\mathbf{max}(c, \mathbf{min}(x, c'))$ or c. In either case, $ob_\alpha(U) = c$.

Example 19 $ob_\alpha(\mu\rho.\uparrow\rho) = \mathbf{lfp}(\lambda x.x + 1) = \mathbf{lfp}(\lambda x.\mathbf{max}(\mathbf{min}(x + 1, \infty))) = \mathbf{lfp}(\lambda x.\mathbf{max}(\infty)) = \infty$.

5.2.3 Reducing $rel(U)$

Next, we eliminate constraints of the form $rel(U)$.

By definition, $rel(U)$ holds if and only if *con*(U') holds for every U' such that $U \longrightarrow^* U'$. The set of such U' can be infinite, but we can normalize U' by using the lemma below. We write $\cong$ for the least equivalence relation on usages that satisfies the monoid laws on $\mid$ (where $\mathbf{0}$ is the unit) and the laws $\uparrow^{(t_1,t_2)}(\uparrow^{(t_1',t_2')}U) \cong \uparrow^{(\mathbf{max}(t_1,t_1'),\mathbf{max}(t_2,t_2'))}U$ and $\uparrow^{(t_1,t_2)}(U_1 \mid U_2) \cong \uparrow^{(t_1,t_2)}U_1 \mid \uparrow^{(t_1,t_2)}U_2$. We write nU for parallel composition of n occurrences of U. $0U$ is $\mathbf{0}$. We say that a usage U is *atomic* if the outermost constructor of U is neither $\mathbf{0}$, $\mid$ nor $\uparrow^{(t_1,t_2)}$.

Lemma 8 *For any U, there exists a finite set of usages $\{U_1, \ldots, U_n\}$ such that for any U' such that $U \longrightarrow^* U'$, there exist $k_1, \ldots, k_n \in \mathbf{Nat}$ such that $U' \cong k_1U_1 \mid \cdots \mid k_nU_n$.*

Proof Investigation of each rule for $U \preceq U'$ and $U \longrightarrow U'$ shows that each atomic usage in U' are introduced by each reduction is either a subformula of U or those obtained by expanding recursion (by UP-REC) or raising the capability level of input prefixes (by UP-↑). Let $\mathbf{Atoms}(U)$ be the set of atomic usages that are sub-expressions of usages obtained by finitely expanding U and do not contain "redundant unfolding" of the form $[\rho \mapsto \mu\rho.U_1]U_1$ as a proper sub-expression (i.e., a sub-expression which is not the expression itself). $\mathbf{Atoms}(U)$ is a finite set. Let **EAtoms**(U) be the set:

$$\begin{array}{l}\mathbf{Atoms}(U) \cup \{\alpha_{t_2}^{\mathbf{max}(t_{11},\ldots,t_{1n})}.U_1 \mid \alpha_{t_2}^{t_1}.U_1 \in \mathbf{Atoms}(U)\} \\ \cup \{\uparrow^{(\mathbf{max}(t_{11},\ldots,t_{1m}),\mathbf{max}(t_{21},\ldots,t_{2n}))}U_1 \\ \qquad \mid U_1 \in \mathbf{Atoms}(U) \text{and } U_1 \text{ is not of the form } \alpha_{t_2}^{t_1}.U\}\end{array}$$

Here, $t_{11},\ldots,t_{1n},t_{21},\ldots,t_{2n}$ ranges over the set of level expressions that occur in U. **EAtoms**(U) is also a finite set. By the above observation, the required property holds for **EAtoms**$(U) = \{U_1,\ldots,U_n\}$. □

For example, if U is $\uparrow^{(3,2)}\mu\rho.(I_1^1.\rho \mid O_2^2)$, **EAtoms**$(U)$ is the set:

$$\begin{array}{l}\{\uparrow^{(t_1,t_2)}\mu\rho.(I_1^1.\rho \mid O_2^2) \mid t_1,t_2 \in \{1,2,3\}\} \\ \cup\{I_{t_2}^{t_1}.\mu\rho.(I_1^1.\rho \mid O_2^2) \mid t_1,t_2 \in \{1,2,3\}\} \\ \cup\{O_{t_2}^{t_1} \mid t_1,t_2 \in \{1,2,3\}\}\end{array}$$

The following lemma ensures that whether *con*(U') holds depends only on whether the indices $k_1,\ldots,k_n$ are 0 or not, so that we need to check only a finite number of cases.

Lemma 9 *If $U' \cong k_1U_1 \mid \cdots \mid k_nU_n$ with $k_1,\ldots,k_n > 0$, then con(U') if and only if con$(U_1 \mid \cdots \mid U_n)$.*

Proof By the definitions of ob_α, cap_α, and $\cong$, (i) if $U \cong U'$ then $ob_\alpha(U) = ob_\alpha U'$ and $cap_\alpha(U) = cap_\alpha(U')$, and (ii) for any $k > 0$, $ob_\alpha(kU) = ob_\alpha(U)$ and $cap_\alpha(kU) = cap_\alpha(U)$. The required property follows immediately follow from those properties. □

By the decidability of the reachability problem of Petri nets [21], we have the following lemma.

Lemma 10 *Let $\{U_1,\ldots,U_m\}$ be a subset of the set of usages in Lemma 8. Then, it is decidable whether there exist $k_1,\ldots,k_m > 0$ such that $U \longrightarrow^* \cong k_1U_1 \mid \cdots \mid k_mU_m$.*

Proof We can reduce the problem into the reachability problem of Petri nets as follows. For each usage expression U_1 in **EAtoms**(U), we prepare two places X_{U_1} and Y_{U_1} of a Petri net. Intuitively, a usage expression $k_1U_1 \mid \cdots k_nU_n$ is encoded into a state of the Petri net $k_1X_{U_1} \mid \cdots \mid k_nX_{U_n}$ (which expresses the state where there are k_i markings in each place X_{U_i}). The place Y_{U_1} is used for testing whether U_1 occurs at the top level.

We prepare the following transitions of the Petri net.

– For each U_1 in **EAtoms**(U), the transitions $X_{U_1} \longrightarrow Y_{U_1}$ and $X_{U_1} \mid Y_{U_1} \longrightarrow Y_{U_1}$.

– Transitions that correspond to each rule for $\preceq$ and $\longrightarrow$.
  For example, for each $I^{t_1}_{t_2}.U_1$ and $O^{t'_1}_{t'_2}.U'_1$ in $\mathbf{EAtoms}(U)$, we add the transition

$$X_{I^{t_1}_{t_2}.U_1}|X_{O^{t_1}_{t_2}.U'_1} \longrightarrow k_1X_{V_1}|\cdots|k_nX_{V_n},$$
$$\text{where } k_1V_1 \mid \cdots \mid k_nV_n \cong U_1 \mid U'_1 \text{ and } V_1,\ldots,V_n \in \mathbf{EAtoms}(U).$$

  For each $U_1 \,\&\, U_2 \in \mathbf{EAtoms}(U)$, we add the transitions:

$$X_{U_1\&U_2} \longrightarrow k_1X_{V_1}|\cdots|k_nX_{V_n}$$
$$\text{where } k_1V_1 \mid \cdots \mid k_nV_n \cong U_1 \text{ and } V_1,\ldots,V_n \in \mathbf{EAtoms}(U)$$
$$X_{U_1\&U_2} \longrightarrow k_1X_{V_1}|\cdots|k_nX_{V_n}$$
$$\text{where } k_1V_1 \mid \cdots \mid k_nV_n \cong U_2 \text{ and } V_1,\ldots,V_n \in \mathbf{EAtoms}(U)$$

Let $U \cong k'_1V_1 \mid \cdots \mid k'_nV_n$ and $V_1,\ldots,V_n,U_1,\ldots,U_m \in \mathbf{EAtoms}(U)$. Then, by the above construction of the Petri net, $U \longrightarrow^* k_1U_1 \mid k_mU_m$ for some $k_1,\ldots,k_m > 0$ if and only if the marking $Y_{U_1}|\cdots|Y_{U_m}$ is reachable from the initial marking $k'_1X_{V_1}|\cdots|k'_nX_{V_n}$. Thus, the problem is decidable [21]. □

Thus, the constraint $rel(U)$ is replaced by the conjunction of constraints of the form $ob_{\overline{\alpha}}(U_1 \mid \cdots \mid U_m) \leq cap_\alpha(U_1 \mid \cdots \mid U_m)$. By the result of the previous subsection, such a constraint can be further reduced to a constraint of the form $t_1 \leq t_2$. Moreover, by the definition of $cap_\alpha(U)$, t_2 can be expressed in the form $\mathbf{min}(\eta_1,\ldots,\eta_n)$ where $\eta_1,\ldots,\eta_n$ are level variables. So, $t_1 \leq t_2$ can be further reduced to $t_1 \leq \eta_1,\ldots,t_1 \leq \eta_n$.

5.2.4 Final step

Now the resulting constraint is a set of constraints of the form:

$$t \leq \eta \quad l_1 \sqsubseteq l_2 \quad \eta = \infty \Rightarrow l_1 \sqsubseteq l_2 \quad l = \mathbf{L} \Rightarrow \eta = \infty$$

Constraints of the first and fourth forms can be solved by a symbolic method. The constraint $l = \mathbf{L} \Rightarrow \eta = \infty$ can be first converted into a constraint $\mathbf{IF}(l = \mathbf{L}, \infty, 0) \leq \eta$ of the first form, where $\mathbf{IF}(e_1,e_2,e_3)$ is e_2 if e_1 holds and it is e_3 otherwise. In each constraint of the first form, t can be normalized to $\mathbf{max}(e_1,\ldots,e_n)$, where each e_i is of the form $\mathbf{min}(\eta+n,c)$ or c, and c does not contain the variable η. Notice that $\mathbf{max}(\mathbf{min}(\eta,c),e) \leq \eta$ if and only if $e \leq \eta$, and that for $n > 0$, $\mathbf{max}(\mathbf{min}(\eta+n,c),e) \leq \eta$ if and only if $\mathbf{max}(c,e) \leq \eta$. So, we can remove η from t in $t \leq \eta$. Thus, we can eliminate the level variable η by substituting t for the occurrences of η in the other constraints.

After all the level variables have been eliminated, the remaining constraints can be normalized to the following set of constraints:

$$\{\delta_1 \sqsubseteq F_1(\delta_1,\ldots,\delta_n),\ \cdots\ \delta_n \sqsubseteq F_n(\delta_1,\ldots,\delta_n),$$
$$l_1 \sqsubseteq G_1(\delta_1,\ldots,\delta_n),\ \cdots,\ l_m \sqsubseteq G_m(\delta_1,\ldots,\delta_n)\}$$

Here, δ_i is a secrecy variable, and l_j is a constant $\mathbf{L}$ or $\mathbf{H}$. F_i and G_j are monotonic functions on $\delta_1,\ldots,\delta_n$, constructed from $\mathbf{H}$, $\mathbf{L}$, secrecy variables, and $\mathbf{IF}(t = \infty, l, \mathbf{H})$ (where l is a secrecy variable or a constant). Note that each constraint of the form $t = \infty \Rightarrow l_1 \sqsubseteq l_2$ has been normalized to $l_1 \sqsubseteq \mathbf{IF}(t = \infty, l_2, \mathbf{H})$. Here, t is anti-monotonic with respect to secrecy variables, hence $\mathbf{IF}(t = \infty, l_2, \mathbf{H})$ is monotonic on secrecy variables.

Let us abbreviate the above constraints to $\{\widetilde{\delta} \sqsubseteq \widetilde{F}(\widetilde{\delta}), \widetilde{l} \sqsubseteq \widetilde{G}(\widetilde{\delta})\}$. Since $\widetilde{F}$ is monotonic, we have:

$\widetilde{\mathbf{H}} \sqsupseteq \widetilde{F}(\widetilde{\mathbf{H}}) \sqsupseteq \widetilde{F}^2(\widetilde{\mathbf{H}}) \sqsupseteq \cdots$

So, we can find $k \in \mathbf{Nat}$ such that $\widetilde{\delta} = \widetilde{F}^k(\widetilde{\mathbf{H}})$ is the greatest solution for $\widetilde{\delta} \sqsubseteq \widetilde{F}(\widetilde{\delta})$. For such k, the constraint $\{\widetilde{\delta} \sqsubseteq \widetilde{F}(\widetilde{\delta}), \widetilde{l} \sqsubseteq \widetilde{G}(\widetilde{\delta})\}$ is satisfiable if and only if $\widetilde{l} \sqsubseteq \widetilde{G}(\widetilde{F}^k(\widetilde{\mathbf{H}}))$ holds.

5.3 Examples

Example 20 Let us consider the following process P:

$(\nu s : \langle\beta\rangle^{\mathbf{H}})\,(\nu z : \langle\,\rangle^{\delta_z})\,(*s(y)\,.\,(\overline{y}\langle\,\rangle \mid \overline{s}\langle y\rangle) \mid \overline{s}\langle z\rangle \mid z(\,).\,\overline{x}\langle\,\rangle)$

To check that $x : \langle\,\rangle^{\mathbf{L}}/\rho_x \vdash_{\mathbf{L}} P$ holds for some β and δ_z, it suffices to compute $Tinf(\Gamma, \mathbf{L}, P)$ for $\Gamma = x : \langle\,\rangle^{\mathbf{L}}/\rho_x$ and check whether it is satisfiable. We list the output of PT for some of the sub-expressions of P.

$$
\begin{array}{l}
PT(*s(y)\,.\,(\overline{y}\langle\,\rangle \mid \overline{s}\langle y\rangle)) = \\
\quad (s : \langle\beta_1\rangle^{\delta_s}/*I^0_{\eta_1}.O^0_{\eta_2}, \delta_0, \{\beta_1 \le \langle\,\rangle^{\delta_y}/O^0_{\eta_4} \mid \uparrow^{(\eta_2+1,\eta_2+1)}\uparrow\beta_1, \\
\qquad \delta_0 \sqsubseteq \delta_s, \delta_0 \sqsubseteq \delta_y, \eta_1 = \infty \Rightarrow \delta_s \sqsubseteq \delta_y\}) \\
PT(\overline{s}\langle z\rangle) = ((z : \uparrow^{(\eta_3+1,\eta_3+1)}\uparrow\beta_2, s : \langle\beta_2\rangle^{\delta_s}/O^0_{\eta_3}), \delta_1, \{\delta_1 \sqsubseteq \delta_s\}) \\
PT(z(\,).\,\overline{x}\langle\,\rangle) = \\
\quad ((x : \uparrow^{(\eta_5+1,\eta_5+1)}\langle\,\rangle^{\delta_x}/O^0_{\eta_6}, z : \langle\,\rangle^{\delta_z}/I^0_{\eta_5}), \delta_2, \\
\quad \{\delta_2 \sqsubseteq \delta_x, \delta_2 \sqsubseteq \delta_z, \eta_5 = \infty \Rightarrow \delta_z \sqsubseteq \delta_x\})
\end{array}
$$

Here, we have omitted unimportant constraints and those of the form $WF(\tau)$: in particular, we have substituted $\mathbf{0}$ for usage variables ρ that can be constrained by only $\rho \le \mathbf{0}$.

$Tinf(\Gamma, \mathbf{L}, P)$ produces the following constraints in addition to the constraints generated for sub-expressions.

$$
\begin{array}{l}
\{\langle\,\rangle^{\delta}/\rho_z = \uparrow^{(\eta_3+1,\eta_3+1)}\uparrow\beta_1 \mid \langle\,\rangle^{\delta_z}/I^0_{\eta_5}, rel(\rho_z), \\
\langle\beta\rangle^{\mathbf{H}}/\rho_s = \langle\beta_1\rangle^{\delta_s}/*I^0_{\eta_1}.O^0_{\eta_2} \mid \langle\beta_2\rangle^{\delta_s}/O^0_{\eta_3}, rel(\rho_s), \\
\langle\,\rangle^{\mathbf{L}}/\rho_x \le \uparrow^{(\eta_5+1,\eta_5+1)}\langle\,\rangle^{\delta_x}/O^0_{\eta_6}, \\
\mathbf{L} \sqsubseteq \delta_0, \delta_1, \delta_2, \\
WF(\langle\beta\rangle^{\mathbf{H}}/\rho_s), WF(\uparrow^{(\eta_5+1,\eta_5+1)}\langle\,\rangle^{\delta_x}/O^0_{\eta_6})\}
\end{array}
$$

By reducing constraints on types, we obtain the following constraints (trivial constraints have been removed):

$$
\begin{array}{l}
\beta = \beta_1 = \beta_2 = \langle\,\rangle^{\delta_z}/\rho_y \\
\rho_z = (\uparrow^{(\eta_3+1,\eta_3+1)}\uparrow\rho_y) \mid I^0_{\eta_5} \\
\rho_s = (*I^0_{\eta_1}.O^0_{\eta_2}) \mid O^0_{\eta_3} \\
\rho_x \le \uparrow^{(\eta_5+1,\eta_5+1)}O^0_{\eta_6} (= O^{\eta_5+1}_{\eta_6}) \\
\rho_y \le O^0_{\eta_4} \mid \uparrow^{(\eta_2+1,\eta_2+1)}\uparrow\rho_y \\
rel(\rho_z) \qquad rel(\rho_s) \\
\delta_s = \mathbf{H} \qquad \delta_x = \mathbf{L} \qquad \eta_5 = \infty \Rightarrow \delta_z \sqsubseteq \delta_x \qquad \eta_1 = \infty \Rightarrow \mathbf{H} \sqsubseteq \delta_z \\
\mathbf{H} \sqsubseteq \delta_z \qquad \eta_6 = \infty
\end{array}
$$

Here, $\rho_s, \rho_x, \rho_y, \rho_z$ are usages of s, x, y, and z respectively. Constraints on the last line come from the conditions on well-formed types. By solving the subusage constraints, we obtain the following solutions for $\rho_s, \rho_x, \rho_y, \rho_z$:

$$\begin{array}{l}\rho_s = (*I^0_{\eta_1}.O^0_{\eta_2}) \mid O^0_{\eta_3}\\ \rho_x = O^{\eta_5+1}_{\eta_6}\\ \rho_y = \mu\rho.(O^0_{\eta_4} \mid \uparrow^{(\eta_2+1,\eta_2+1)}\uparrow\rho)\\ \rho_z = (\uparrow^{(\eta_3+1,\eta_3+1)}\uparrow\rho_y) \mid I^0_{\eta_5}\\ \quad = (\uparrow^{(\eta_3+1,\eta_3+1)}\uparrow\mu\rho.(O^0_{\eta_4} \mid \uparrow^{(\eta_2+1,\eta_2+1)}\uparrow\rho)) \mid I^0_{\eta_5}\end{array}$$

To remove $\uparrow$, we compute $ob_I(\rho_y)$ and $ob_O(\rho_y)$.

$$\begin{array}{ll}ob_I(\rho_y) & = \mathbf{lfp}(\lambda x.\mathbf{min}(\infty, \mathbf{max}(\eta_2+1, x+1)))\\ & = \mathbf{lfp}(\lambda x.\mathbf{max}(\eta_2+1, \mathbf{min}(x+1, \infty))\\ & = \mathbf{lfp}(\lambda x.\mathbf{max}(\eta_2+1, \infty)) = \infty\\ ob_O(\rho_y) & = \mathbf{lfp}(\lambda x.\mathbf{min}(0, \mathbf{max}(\eta_2+1, x+1)))\\ & = \mathbf{lfp}(\lambda x.0) = 0\end{array}$$

So, ρ_y and ρ_z are: $\mu\rho.(O^0_{\eta_4} \mid \uparrow^{(\infty,\eta_2+1)}\rho)$ and $(\uparrow^{(\infty,\eta_3+1)}\rho_y) \mid I^0_{\eta_5}$. Since $\rho_s \longrightarrow \rho_s \longrightarrow \cdots$, $cap_I(\rho_s) = \eta_1$ and $cap_O(\rho_s) = \mathbf{min}(\eta_2, \eta_3)$, $rel(\rho_1)$ is reduced to the constraints $0 \leq \mathbf{min}(\eta_2, \eta_3)$ and $0 \leq \eta_1$ (which are tautologies). On the other hand, ρ_z can be reduced to $U_1 = \uparrow^{(\infty,\mathbf{max}(\eta_3+1,\eta_2+1))}\rho_3$ or $U_2 = O^{\eta_3+1}_{\eta_4} \mid U_1$. So, $rel(\rho_z)$ can be reduced to

$$\begin{array}{ll}ob_I(\rho_z) \leq cap_O(\rho_z) & ob_O(\rho_z) \leq cap_I(\rho_z)\\ ob_I(U_i) \leq cap_O(U_i) & ob_O(U_i) \leq cap_I(U_i)\ (i = 1, 2),\end{array}$$

from which we obtain $\eta_3 + 1 \leq \eta_5$ and $\infty \leq \eta_4$. (We have omitted tautology). So, the remaining constraints are:

$$\begin{array}{lll}\infty \leq \eta_6 & \eta_3 + 1 \leq \eta_5 & \infty \leq \eta_4\\ \mathbf{H} \sqsubseteq \delta_z & l \sqsubseteq \mathbf{IF}(\eta_5 = \infty, \mathbf{L}, \mathbf{H}) & \end{array}$$

The least solution for constraints on level variables is: $\eta_1 = \eta_2 = \eta_3 = 0, \eta_5 = \eta_3 + 1 = 1, \eta_4 = \eta_6 = \infty$. So, we obtain the constraints $\{\mathbf{H} \sqsubseteq \delta_z, \delta_z \sqsubseteq \mathbf{H}\}$, which hold for $\delta_z = \mathbf{H}$.

Example 21 Let P be the process:

$$(\nu x : \langle\,\rangle^{\mathbf{H}})\,(\nu secret : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}})\,(A \mid C \mid \overline{x}\langle\,\rangle \mid \overline{secret}\langle b \rangle)$$

in Example 15 and Γ be $public : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}} / \rho_1, x : \langle\,\rangle^{\mathbf{L}} / \rho_2$. By computing $Tinf(\Gamma, \mathbf{L}, P)$ and reducing constraints on types, we obtain the following constraints.

$\rho_1 \leq \uparrow^{(\eta_5+1,\eta_5+1)} \uparrow^{(\eta_{15}+1,\eta_{15}+1)} I^0_{\eta_1}.O^0_{\eta_2}$
$\rho_2 \leq I^0_{\eta_3}.\uparrow^{(\eta_{10}+1,\eta_{10}+1)} O^0_{\eta_4}$
$\rho_3 \leq I^0_{\eta_5}.O^0_{\eta_6} \mid O^0_{\eta_7}$
$\rho_4 \leq \uparrow^{(\eta_5+1,\eta_5+1)} (I^0_{\eta_8}.O^0_{\eta_9} \,\&\, \mathbf{0}) \mid \uparrow^{(\eta_3+1,\eta_3+1)} I^0_{\eta_{10}}.O^0_{\eta_{11}} \mid O^0_{\eta_{12}}$
$\rho_5 \leq \uparrow^{(\eta_5+1,\eta_5+1)} ((\uparrow^{(\eta_8+1,\eta_8+1)} O^0_{\eta_{13}} \,\&\, O^0_{\eta_{14}}) \mid I^0_{\eta_{15}})$
$rel(\rho_3) \qquad rel(\rho_4) \qquad rel(\rho_5)$
$\mathbf{Cap}_\infty(\rho_1) \qquad \mathbf{Cap}_\infty(\rho_2) \qquad \delta = \mathbf{L} \Rightarrow \mathbf{Cap}_\infty(\rho_5)$
$\eta_5 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L}$
$\eta_8 = \infty \Rightarrow \mathbf{H} \sqsubseteq \delta$
$\eta_{15} = \infty \Rightarrow \delta \sqsubseteq \mathbf{L}$
$\eta_{10} = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L}$
$\mathbf{H} \sqsubseteq \delta$

Here, ρ_3, ρ_4, ρ_5 are usages of channels $secret, x$, and c respectively, and δ is the secrecy level of c.

By solving the subusage constraints, we obtain:

$\rho_1 = I^{\mathbf{max}(\eta_5+1,\eta_{15}+1)}_{\eta_1}.O^0_{\eta_2}$
$\rho_2 = I^0_{\eta_3}.O^{\eta_{10}+1}_{\eta_4}$
$\rho_3 = I^0_{\eta_5}.O^0_{\eta_6} \mid O^0_{\eta_7}$
$\rho_4 = (I^{\eta_5+1}_{\eta_8}.O^0_{\eta_9} \,\&\, \mathbf{0}) \mid I^{\eta_3+1}_{\eta_{10}}.O^0_{\eta_{11}} \mid O^0_{\eta_{12}}$
$\rho_5 = (O^{\mathbf{max}(\eta_5+1,\eta_8+1)}_{\eta_{13}} \,\&\, O^{\eta_5+1}_{\eta_{14}}) \mid I^{\eta_5+1}_{\eta_{15}}$

Therefore, the constraints:

$rel(\rho_3) \qquad rel(\rho_4) \qquad rel(\rho_5)$
$\mathbf{Cap}_\infty(\rho_1) \qquad \mathbf{Cap}_\infty(\rho_2) \qquad \delta = \mathbf{L} \Rightarrow \mathbf{Cap}_\infty(\rho_5)$

are reduced to:

$\infty \leq \eta_6$
$\eta_3 + 1 \leq \eta_{12} \qquad \infty \leq \eta_{11} \qquad \infty \leq \eta_9$
$\eta_5 + 1 \leq \eta_{13} \qquad \eta_5 + 1 \leq \eta_{14} \qquad \mathbf{max}(\eta_5 + 1, \eta_8 + 1) \leq \eta_{15}$
$\infty \leq \eta_1, \eta_2, \eta_3, \eta_4$
$\mathbf{IF}(\delta = \mathbf{L}, \infty, 0) \leq \eta_{13}, \eta_{14}, \eta_{15}$

By solving them, we obtain the following solution for level variables:

$\eta_1 = \eta_2 = \eta_3 = \eta_4 = \eta_6 = \eta_9 = \eta_{11} = \eta_{12} = \infty$
$\eta_5 = \eta_7 = \eta_8 = \eta_{10} = 0$
$\eta_{13} = \eta_{14} = \eta_{15} = \mathbf{max}(1, \mathbf{IF}(\delta = \mathbf{L}, \infty, 0))$

By substituting it for the constraints on secrecy levels, we obtain:

$0 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L}$
$0 = \infty \Rightarrow \mathbf{H} \sqsubseteq \delta$
$\mathbf{max}(1, \mathbf{IF}(\delta = \mathbf{L}, \infty, 0)) = \infty \Rightarrow \delta \sqsubseteq \mathbf{L}$
$0 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L}$
$\mathbf{H} \sqsubseteq \delta$

They are normalized to the following constraints:

$\mathbf{H} \sqsubseteq \mathbf{IF}(0 = \infty, \mathbf{L}, \mathbf{H})$
$\mathbf{H} \sqsubseteq \mathbf{IF}(0 = \infty, \delta, \mathbf{H})$
$\delta \sqsubseteq \mathbf{IF}(\mathbf{max}(1, \mathbf{IF}(\delta = \mathbf{L}, \infty, 0)) = \infty, \mathbf{L}, \mathbf{H})$
$\mathbf{H} \sqsubseteq \mathbf{IF}(0 = \infty, \mathbf{L}, \mathbf{H})$
$\mathbf{H} \sqsubseteq \delta$

The constraint is satisfied for $\delta = \mathbf{H}$.

5.4 Implementation

We have implemented an information flow analyzer `TyPiCal`, which is available from `http://www.kb.ecei.tohoku.ac.jp/~koba/typical/`. The implementation is mostly based on the algorithm described above, but there are the following discrepancies between the above algorithm and the current implementation of `TyPiCal`.

- `TyPiCal` allows *no* annotation on secrecy levels. `TyPiCal` simply tries to assign the level $\mathbf{H}$ to as many values as possible.
- `TyPiCal`TyPiCal allows no declaration on the environment. `TyPiCal`TyPiCal assumes that all the free channels are low-level channels.
- `TyPiCal` allows subtyping on base types and pair type constructors (see Section 6) to enhance the analysis.
- `TyPiCal` uses a sound but *incomplete* algorithm for the Petri net reachability problem, so the current implementation is not complete with respect to the type system described in this paper.

The first and second are just limitations of the current interface, not those of the core of the implemented algorithm. The last one is inevitable as no complete, *efficient* algorithm for the Petri net reachability problem is known. To obtain an approximated solution for the Petri net reachability problem, we abstract a Petri net into a finite state machine, and then solve the reachability problem for the finite state machine by using boolean decision diagrams. The detail will be described in a separate technical report.

Figure 5 shows a sample input for `TyPiCal`. The code is based on the bank account example given in Example 7. The current balance (100) is kept in the message `balance!100` on the second line. The last but two lines expresses the client process given at the end of Example 7 (where we have removed the first input from y so that `y!withdraw` serves as an initializer for the shared variable). The last two lines simulate high-level client processes which may access *withdraw* and *deposit* methods an arbitrary number of times. The only free name is *transfer*. So, low-level processes are assumed to access only through that name. Figure 6 shows a sample session for the sample code.[6] As the figure shows, `TyPiCal` outputs the code whose base values (booleans and integers) are annotated with secrecy levels. Note that the current balance (100) is annotated with `/*H*/`, which

[6] `TyPiCal` can also perform other kinds of analyses; the option `-i` specifies that the information flow analyzer should be invoked. We have modified indentations and line breaks of the actual output for the clarity.

```
new balance in new withdraw in new getBalance in new y in
  balance!100 |
  *withdraw?z.(let amount=fst(z) in let r = snd(z) in
      balance?x.
        if x>=amount then (r!true | balance!(x-amount))
        else (r!false | balance!x)) |
  *getBalance?r.balance?x.(r!x | balance!x) |
  *deposit?z.(let amount=fst(z) in let r = snd(z) in
      if amount>=0
      then balance?x.(r!true | balance!(x+amount))
      else r!false) |
  y!withdraw | y?m.(y!m | new r in m!(1,r)) |
  *(if true then (new r in getBalance!r) else 0) |
  *(if true then (new r in withdraw!(1,r)) else 0)
```

Fig. 5 A sample input for `TyPiCal`: account.pi

```
% typical -i account.pi
TyPiCal 1.0.2: A Type-based static analyzer for the Pi-...
analyzing account.pi...
new balance in new withdraw in new getBalance in new y in
  balance!100/*H*/ |
  *withdraw?z.(let amount=fst(z) in let r = snd(z) in
      balance?x.
        if x>=amount then (r!true/*H*/ | balance!(x-amount))
        else (r!false/*H*/ | balance!x)) |
  *getBalance?r.balance?x.(r!x | balance!x) |
  *deposit?z.(let amount=fst(z) in let r = snd(z) in
      if amount>=0/*L*/
      then balance?x.(r!true/*L*/ | balance!(x+amount))
      else r!false/*L*/) |
  y!withdraw | y?m.(y!m | new r in m!(1/*H*/,r)) |
  *(if true/*H*/ then (new r in getBalance!r) else 0) |
  *(if true/*H*/ then (new r in withdraw!(1/*H*/,r)) else 0)
Elapsed Time: 0.16sec
```

Fig. 6 The output of `TyPiCal` for account.pi

means that the balance is kept secret to the environment, which can access the account only through *transfer*.

Figure 7 shows an unsafe version of the account code. Here, the current balance is returned as the result of the *transfer* method. `TyPiCal` annotates the current balance 100 with `/*L*/` for this example, which means that information about the current balance may be leaked to the environment.

6 Discussions

In Section 3, we have imposed several restrictions on the type system for the sake of simplicity. In this section, we discuss those restrictions and how to remove them.

*Conditions on well-formed types*In Section 3.1, we have presented IN-IDEAL as an ideal rule for input processes. Actually, however, we also need some subtle,

```
new balance in new withdraw in new getBalance in new y in
  balance!100 |
  *withdraw?z.(let amount=fst(z) in let r = snd(z) in
      balance?x.
        if x>=amount then (r!true | balance!(x-amount))
        else (r!false | balance!x)) |
  *getBalance?r.balance?x.(r!x | balance!x) |
  *transfer?z.(let amount=fst(z) in let r = snd(z) in
      if amount>=0
      then balance?x.(r!(x+amount) | balance!(x+amount))
      else r!0) |
  *(if true then (new r in getBalance!r) else 0) |
  *(if true then (new r in withdraw!(1,r)) else 0)
```

Fig. 7 An unsafe account

extra conditions, which are implicitly enforced by the conditions on well-formed types (Definition 14): If x is a high-level channel and P is a low-level process, $x(y).\,P$ is safe only if the success of x is guaranteed by only communications on high-level channels and if y is not used as a low-level value in P. Indeed, without the assumption about well-formed types, the typing rules in Section 3 are unsound. For example, consider the following process (which violates the first condition on well-formed types):

$$(\nu x : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{H}})$$
$$(\mathbf{if}\ \mathit{secret}\ \mathbf{then}\ \overline{x}\langle \mathit{true}^{\mathbf{L}} \rangle\ \mathbf{else}\ \overline{x}\langle \mathit{false}^{\mathbf{L}} \rangle$$
$$\mid x(y).\,\overline{\mathit{non\_secret}}\langle y \rangle)$$

It is well-typed under the type environment:

$$\mathit{secret} : \mathbf{bool}^{\mathbf{H}}, \mathit{non\_secret} : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}} / O_{\infty}^{\infty}$$

if we remove the first condition on well-formed types, but it leaks the value of secret.

Let us also consider the following process:

$$(\nu x : \langle\,\rangle^{\mathbf{H}})\,(\nu y : \langle \mathit{int}^{\mathbf{L}} \rangle^{\mathbf{L}})$$
$$(\mathbf{if}\ \mathit{secret}\ \mathbf{then}\ \overline{x}\langle\,\rangle\ \mathbf{else}\ \mathbf{0}$$
$$\mid \overline{y}\langle \mathit{true}^{\mathbf{L}} \rangle \mid x(\,).\,\overline{y}\langle \mathit{false}^{\mathbf{L}} \rangle$$
$$\mid y(z).\,(\overline{x}\langle\,\rangle \mid \overline{\mathit{non\_secret}}\langle z \rangle))$$

It is well typed under the type environment:

$$\mathit{secret} : \mathbf{bool}^{\mathbf{H}}, \mathit{non\_secret} : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}} / O_{\infty}^{\infty}$$

if we remove the second condition on well-formed types. (Let the usage of y be $O_0^0 \mid O_\infty^\infty \mid I_0^0$ and the usage of x be $(O_\infty^\infty \,\&\, \mathbf{0}) \mid I_1^\infty \mid O_\infty^1$.) The process, however, leaks information about secret, since $\mathit{false}^{\mathbf{L}}$ can be sent on $\mathit{non\_secret}$ only if secret is $\mathit{true}^{\mathbf{H}}$.

One way to remove the first condition on well-formed types would be to replace the condition $t_c = \infty \Rightarrow l_1 \sqsubseteq l_2$ in rule T-IN with $(t_c = \infty \vee \mathbf{SL}(\widetilde{\tau}) = \mathbf{L}) \Rightarrow l_1 \sqsubseteq l_2$, where $\mathbf{SL}(\widetilde{\tau})$ is the least secrecy level annotation that appears in $\widetilde{\tau}$. (We have not yet checked whether this is a sufficient condition.) Then, in the first example above, x must be low-level, so that $\mathbf{if}\ \mathit{secret}\ \mathbf{then}\ \overline{x}\langle \mathit{true}^{\mathbf{L}} \rangle\ \mathbf{else}\ \overline{x}\langle \mathit{false}^{\mathbf{L}} \rangle$

is ill-typed. The erasure function $\mathbf{Er}$ must be redefined, so that a high-level communication is not erased if it carries low-level values.

The second condition on well-formed types can be removed by changing the operation $\uparrow^{(t_c+1,t_c+1)}$ in T-OUT and T-IN so that if the channel x is low-level, then obligation levels of high-level channels are raised to ∞. Note that the problem of the second example above was that the output on the high-level channel x on the last line being executed depended on the input capability on the low-level channel y.

*Subtyping*Our type system does not allow subtyping based on secrecy levels (e.g., $\mathbf{bool}^{\mathbf{L}} \not\leq \mathbf{bool}^{\mathbf{H}}$). Allowing the relation $\mathbf{bool}^{\mathbf{L}} \leq \mathbf{bool}^{\mathbf{H}}$ does not cause any problem: we only need to redefine the erasure function $\mathbf{Er}_\Gamma(P)$, so that, for example, $\mathit{true}^{\mathbf{L}}$ is also replaced by $\star$ if it is used as a value of type $\mathbf{bool}^{\mathbf{H}}$. On the other hand, introducing subtyping on channel types is tricky. In fact, allowing $\langle\widetilde{\tau}\rangle^{\mathbf{L}}/U \leq \langle\widetilde{\tau}\rangle^{\mathbf{H}}/U$ for an arbitrary U makes the type system unsound [27]. Honda and Yoshida's type system [11] allows subtyping based on secrecy levels for certain kinds of channels. It is left for future work to find a general condition on U for $\langle\widetilde{\tau}\rangle^{\mathbf{L}}/U \leq \langle\widetilde{\tau}\rangle^{\mathbf{H}}/U$ to be valid.

Introducing subtyping based on input/output modes [25] does not cause any problem and the type inference algorithm can be developed along the lines of [12].

*Other data structures*The formalization in this paper deals with only booleans as primitive values. We believe that it is straightforward to extend our analysis to deal with other data structures such as integers, pairs, variants, and lists. Indeed, the current implementation of `TyPiCal` already supports integers and pairs (recall Section 5.4).

*Encoding of functions*Unlike Honda and Yoshida's type system [11], there are certain terms of the simply-typed λ-calculus whose termination property cannot be captured in our type system. (For example, our type system cannot guarantee that the process obtained by the call-by-value encoding of $\mathbf{let}\ g = \lambda f.f(1)\ \mathbf{in}\ g(\lambda y.g(\lambda x.x))$, which calls g inside g, will eventually return a result.) To remove the limitation, we need to introduce some form of polymorphism on capability/obligation levels. Alternatively, we can treat functions as primitives and give typing rules for them directly.

*Treatment of shared variables*As discussed in Example 4, we can encode operations on shared variables into communication primitives, but the resulting analysis is not precise enough. For example, consider the following command [10]: `if secret>0 then secret := non_secret`. Since reading from the non-secret variable `non_secret` does not leak any information, the command should be safe. However, its encoding into the π-calculus performs communications on non-secret channels, so that it is judged to be unsafe. To overcome this limitation, it seems necessary to take $x(y).(\overline{x}\langle y\rangle \mid P)$ as a primitive, and give a

special typing rule for it.[7] With such special treatment, we believe that our analysis for shared variables is comparable to other analyses [10, 11].

*Timing leaks*Our type system does not prevent leakage of secret information from the timing behavior. To prevent such leakage, we can use the type system for time-boundedness [14], which can statically guarantee that each communication succeeds in a certain number of reduction steps. An alternative way to avoid timing leaks would be to impose a restriction that programs must be confluent [36].

*Ill-typed contexts*In Section 4, we considered only well-typed contexts as observers (recall Definition 24). In practice, however, we may not be able to assume that malicious processes respect types. To make the non-interference property hold also in the presence of ill-typed processes, we need to extend our type system with a special type for describing untyped values [1, 4, 15].

*More precise analysis for dependencies between different channels*In the type system presented in this paper, dependencies between different channels are controlled only through capabilities/obligations, which is sometimes too restrictive. As discussed elsewhere [14], we can extend our type system using the idea of generic types [13] or graph types [34].

7 Related Work

We have already discussed previous studies on information flow analysis for concurrent programs in Section 1. We discuss some of them in more detail below. Mantel and Sabelfeld [29] have also proposed a type-based information flow analysis for a multi-threaded language with communication primitives. Their analysis suffers from the same problem as Pottier's type system [27] discussed in Section 1. To improve the expressive power, they instead introduced encrypted channels (in addition to high-level/low-level channels) and more communication primitives (such as a primitive for non-blocking receive). Such a solution is orthogonal to our approach, so that we can combine them to obtain a more expressive secure concurrent language. Hennessy and Riely [8, 9] have also studied a type system for the asynchronous π-calculus. Their type system also suffers from the same problem as Pottier's one.

The idea of refining secrecy analysis based on information about the success of synchronization seems to go back to the idea of linear continuations of Zdancewic and Myers for a sequential language with continuations [35]. Along that line of work, Zdancewic and Myers [36] have recently proposed a type system for a concurrent language having (a restricted form) of join-patterns [6] as synchronization primitives. To overcome the problem discussed in Section 1, their type system introduce linear (use-once) channels and control the temporal ordering on communications on linear channels (which are controlled in our

[7] This is against our goal to uniformly treat various concurrency primitives, but it is inevitable since there is no way to distinguish between $x(n).\,(\overline{x}\langle n\rangle \mid P)$ and $x(n).\,(\overline{x}\langle n+1\rangle \mid P)$ at the type-level: Note that if x is a non-secret channel, the former does not leak any information in the asynchronous π-calculus, while the latter does.

type system through capability/obligation levels) in a syntactic manner. It seems fairly easy to encode their typed calculus into our typed calculus (although extensions with subtyping discussed in Section 6 are necessary). In fact, a join-pattern $\mathbf{let}\ c_1(\widetilde{x_1})|\cdots|c_n(\widetilde{x_n}) \triangleright P\ \mathbf{in}\ Q$ can be encoded into

$$(\nu c_1)\ \cdots (\nu c_n)\ (*c_1(\widetilde{x_1}).\ \cdots c_n(\widetilde{x_n}).\ P \mid Q)$$

and a linear join-pattern $\mathbf{let}\ c_1(\widetilde{x_1})|\cdots|c_n(\widetilde{x_n}) \multimap P\ \mathbf{in}\ Q$ can be encoded into

$$(\nu c_1)\ \cdots (\nu c_n)\ (c_1(\widetilde{x_1}).\ \cdots c_n(\widetilde{x_n}).\ P \mid Q)$$

On the other hand, it is not clear how to extend Zdancewic and Myers [36]'s type system to encode our calculus into their calculus. For example, since the success of communications is guaranteed only on linear channels, their type system cannot deal with the example given in Section 1. Moreover, their type system impose restrictions that linear channels cannot be passed through linear channels, etc. (The type system of Honda et al. [10] also imposes similar restrictions.)

Some of the ideas found in Honda and Yoshida's work [11] (such as subtyping on channel types discussed in the previous section) are missing in our type system, so that our type system does not completely subsume their type system. It would be interesting to study how the missing features can be integrated into our type system.

Our type system has been obtained by simplifying and refining our previous type system for lock-freedom [14]. The most important technical contribution with respect to the previous work (besides the extension to deal with secrecy) is the development of a sound and complete type inference algorithm (and refinement of the type system to enable the type inference). The algorithm has been inspired from our type inference algorithm for a deadlock-free π-calculus [18]. The latter algorithm was, however, incomplete (see [18] for details). The completeness of the type inference algorithm in this paper has been obtained by careful definitions of the semantics of recursive usages: The key property of recursive usages is that $\mu\rho.U$ is the greatest usage that satisfies $\rho \leq U$.

Our technique for proving non-interference using the erasure function has been inspired from our recent work on type-based useless-code elimination for the π-calculus [15] and is probably also related with Pottier's proof technique [27]. The proof in the present paper is, however, more sophisticated since we need a lock-freedom property to show the correspondence between P and $\mathbf{Er}_\Gamma(P)$.

8 Conclusion

We have presented a type system for information flow analysis for the pi-calculus and proved its soundness. Like recent type systems for information flow analysis [11, 36], our type system takes into account information that certain communications eventually succeed. Thanks to the uniform treatment of communication/synchronization patterns, our type system can perform more precise analysis than previous type systems for certain communication/synchronization patterns (like synchronization using locks). The uniform treatment also enabled development of a sound and complete type inference algorithm. The result on the sound and complete type inference algorithm also serves as a refinement of our previous

work on deadlock/livelock-freedom [14, 18, 32] We have implented a prototype analysis tool `TyPiCal` [16] based on the result described in this article.

Our type system in this paper can also be used for program slicing and useless-code elimination for concurrent programs, since both information flow analysis and program slicing are instances of dependency analysis [2]. Our previous type system for slicing and useless-code elimination for the π-calculus [15] did not take the lock-freedom property into account, so that it was not so effective. We can refine it by using the type system in the present paper.

Extending our type-based information flow analysis to deal with encryption/decryption primitives [3] is also interesting future work.

Acknowledgements We would like to thank Steve Zdancewic, Martín Abadi, Eijiro Sumii, Lucian Wischik for useful comments and discussions on an earlier version of the paper. We would also like to thank anonymous referees for a number of useful comments.

References

1. Abadi, M.: Secrecy by typing in security protocols. Journal of the Association for Computing Machinery (JACM) **46**(5), 749–786 (1999)
2. Abadi, M., Banerjee, A., Heintze, N., Rieck, J.G.: A core calculus of dependency. In: Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages, pp. 147–169 (1999)
3. Abadi, M., Gordon, A.D.: A Calculus for Cryptographic Protocols: The Spi Calculus. Information and Computation **148**(1), 1–70 (1999)
4. Cardelli, L., Ghelli, G., Gordon, A.D.: Secrecy and group creation. In: Proceedings of CONCUR 2000, *Lecture Notes in Computer Science*, vol. 1877, pp. 365–379. Springer-Verlag (2000)
5. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Communications of the ACM **20**(7), 504–513 (1977)
6. Fournet, C., Gonthier, G.: The reflexive CHAM and the join-calculus. In: Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages, pp. 372–385 (1996)
7. Heintze, N., Riecke, J.: The slam calculus: programming with secrecy and integrity. In: Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages, pp. 365–377 (1998)
8. Hennessy, M.: The security picalculus and non-interference. Journal of Logic and Algebraic Programming (2003). To appear
9. Hennessy, M., Riely, J.: Information flow vs. resource access in the information asynchronous pi-calculus. In: Proceedings of ICALP 2000, *Lecture Notes in Computer Science*, vol. 1853. Springer-Verlag (2000)
10. Honda, K., Vasconcelos, V., Yoshida, N.: Secure information flow as typed process behaviour. In: Proc. of European Symposium on Programming (ESOP) 2000, *Lecture Notes in Computer Science*, vol. 1782, pp. 180–199. Springer-Verlag (2000)
11. Honda, K., Yoshida, N.: A uniform type structure for secure information flow. In: Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages, pp. 81–92 (2002)
12. Igarashi, A., Kobayashi, N.: Type reconstruction for linear pi-calculus with I/O subtyping. Information and Computation **161**, 1–44 (2000)
13. Igarashi, A., Kobayashi, N.: A generic type system for the pi-calculus. Theoretical Computer Science **311**(1-3), 121–163 (2004)
14. Kobayashi, N.: A type system for lock-free processes. Information and Computation **177**, 122–159 (2002)
15. Kobayashi, N.: Useless-code elimination and program slicing for the pi-calculus. In: Proceedings of The First Asian Symposium on Programming Languages and Systems (APLAS'03), *Lecture Notes in Computer Science*, vol. 2895, pp. 55–72 (2003)

16. Kobayashi, N.: TyPiCal: A Type-Based Analyzer for the Pi-Calculus (2004). http://www.kb.ecei.tohoku.ac.jp/~koba/typical/
17. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the pi-calculus. ACM Transactions on Programming Languages and Systems **21**(5), 914–947 (1999)
18. Kobayashi, N., Saito, S., Sumii, E.: An implicitly-typed deadlock-free process calculus. Tech. Rep. TR00-01, Dept. Info. Sci., Univ. of Tokyo (2000). Available from http://www.kb.cs.titech.ac.jp/~kobayasi/. A summary has appeared in Proceedings of CONCUR 2000, Springer LNCS1877, pp.489-503, 2000
19. Kobayashi, N., Shirane, K.: Type-based information flow analysis for a low-level language. Computer Software **20**(2), 2–21 (2003). In Japanese. A summary written in English is available from http://www.kb.cs.titech.ac.jp/~kobayasi/
20. Kobayashi, N., Yonezawa, A.: Towards foundations for concurrent object-oriented programming – types and language design. Theory and Practice of Object Systems **1**(4), 243–268 (1995)
21. Mayr, E.W.: An algorithm for the general petri net reachability problem. SIAM Journal on Computing **13**(3), 441–461 (1984)
22. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
23. Milner, R.: The polyadic π-calculus: a tutorial. In: F.L. Bauer, W. Brauer, H. Schwichtenberg (eds.) Logic and Algebra of Specification. Springer-Verlag (1993)
24. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press (1999)
25. Pierce, B., Sangiorgi, D.: Typing and subtyping for mobile processes. Mathematical Structures in Computer Science **6**(5), 409–454 (1996)
26. Pierce, B.C., Turner, D.N.: Concurrent objects in a process calculus. In: Theory and Practice of Parallel Programming (TPPP), Sendai, Japan (Nov. 1994), *Lecture Notes in Computer Science*, vol. 907, pp. 187–215. Springer-Verlag (1995)
27. Pottier, F.: A simple view of type-secure information flow in the π-calculus. In: Proceedings of the 15th IEEE Computer Security Foundations Workshop, pp. 320–330 (2002)
28. Pottier, F., Simonet, V.: Information flow inference for ML. In: Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages, pp. 319–330 (2002)
29. Sabelfeld, A., Mantel, H.: Static confidentiality enforcement for distributed programs. In: Proceedings of the 9th International Static Analysis Symposium, LNCS 2477, pp. 376–394. Springer-Verlag, Madrid, Spain (2002)
30. Sangiorgi, D., Walker, D.: The Pi-Calculus: A Theory of Mobile Processes. Cambridge University Press (2001)
31. Smith, G., Volpano, D.: Secure information flow in a multi-threaded imperative language. In: Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages, pp. 355–364 (1998)
32. Sumii, E., Kobayashi, N.: A generalized deadlock-free process calculus. In: Proc. of Workshop on High-Level Concurrent Language (HLCL'98), *ENTCS*, vol. 16(3), pp. 55–77 (1998)
33. Volpano, D., Smith, G., Irvine, C.: A sound type system for secure flow analysis. Journal of Computer Security **4**(3), 167–187 (1996)
34. Yoshida, N.: Graph types for monadic mobile processes. In: FST/TCS'16, *Lecture Notes in Computer Science*, vol. 1180, pp. 371–387. Springer-Verlag (1996)
35. Zdancewic, S., Myers, A.C.: Secure information flow via linear continuations. Higher-Order and Symbolic Computation **15**(2/3), 209–234 (2002)
36. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: Proceedings of the 16th IEEE Computer Security Foundations Workshop (2003)

A Proof of Theorem 1

Lemma 11 *1. If $U \preceq U'$, then $U \leq U'$.*
2. $U \mid \mathbf{0} \leq U$.
3. If $ob(U) = \infty$, then $U \leq \mathbf{0}$.
4. If $t_o < ob(U_2)$ and $t_c < ob(U_2)$, then $\alpha^{t_o}_{t_c}.U_1 \mid U_2 \leq \alpha^{t_o}_{t_c}.(U_1 \mid U_2)$.
5. If $ob(U) = \infty$, then $U' \mid U \leq U'$ for any U'.

6. *If* $t'_o \leq t_o$ *and* $t_c \leq t'_c$, *then* $\alpha^{t_o}_{t_c}.U \leq \alpha^{t'_o}_{t'_c}.U$ *holds.*
7. *If* $U_1 \leq [\rho \mapsto U_1]U$, *then* $U_1 \leq \mu\rho.U$.
8. $\uparrow^{(t_1,t_2)}U \leq U$.

Before proving the above lemma, we introduce an "up-to" technique which is often used for proving simulation/bisimulation of processes.

Lemma 12 *Let* $\mathcal{R}$ *be a binary relation on usages. If the following conditions are satisfied for every pair* $(U_1, U_2) \in \mathcal{R}$, $\mathcal{R} \subseteq \leq$.

1. $[\rho \mapsto U_1]U\mathcal{R}[\rho \mapsto U_2]U$ *for any usage* U *such that* $FV(U) = \{\rho\}$.
2. *If* $U_2 \longrightarrow U'_2$, *then there exists* U'_1 *such that* $U_1 \longrightarrow U'_1$ *and* $U'_1 \leq \mathcal{R} \leq U'_2$.
3. *For each* $\alpha \in \{I, O\}$, $cap_\alpha(U_1) \leq cap_\alpha(U_2)$ *holds.*
4. *For each* $\alpha \in \{I, O\}$, *if* $con_{\overline{\alpha}}(U_1)$, *then* $ob_\alpha(U_1) \geq ob_\alpha(U_2)$.

Proof It suffices to show that $\mathcal{R}\cup \leq \mathcal{R} \leq$ satisfies the four conditions in Definition 11. We only show the second condition, as the other conditions are trivial.

Suppose that $(U_1, U_2) \in \leq \mathcal{R} \leq$ and $U_2 \longrightarrow U'_2$. Then there exists U_3 and U_4 such that $U_1 \leq U_3\mathcal{R}U_4 \leq U_2$. By the definition of $\leq$, there exists U'_4 such that $U'_4 \leq U'_2$ and $U_4 \longrightarrow U'_4$. By the condition on $\mathcal{R}$, there exists U'_3 such that $U'_3 \leq \mathcal{R} \leq U'_4$ and $U_3 \longrightarrow U'_3$. Again by the definition of $\leq$, we have U'_1 such that $U'_1 \leq U'_3$ and $U_1 \longrightarrow U'_1$. Therefore, we have $U'_1 \leq \leq \mathcal{R} \leq \leq U'_2$. Since $\leq$ is transitive, $U'_1 \leq \mathcal{R} \leq U'_2$ holds as required.

We now prove Lemma 11.

Proof of Lemma 11 We show only the seventh law, as it is most complex and important. The other laws can be proved in a similar manner. The fifth law follows from the first and second laws by: $U' \mid U \leq U' \mid \mathbf{0} \leq U'$.

Suppose $U_1 \leq [\rho \mapsto U_1]U$ and let $\mathcal{R}$ be $\{([\rho' \mapsto U_1]U_0, [\rho' \mapsto \mu\rho.U]U_0) \mid \{\rho'\} = FV(U_0)\}$. It suffices to show that $\mathcal{R}$ satisfies the conditions of Lemma 12. The first condition is trivial.

To show the second condition, suppose that $[\rho' \mapsto \mu\rho.U]U_0 \longrightarrow U'$. We show that there exists U'' such that $[\rho' \mapsto U_1]U_0 \longrightarrow U''$ and $U'' \leq \mathcal{R} \leq U'$ by induction on the number n of applications of the expansion rule $\mu\rho.U \preceq [\rho \mapsto \mu\rho.U]U$ before the reduction in the derivation of $[\rho' \mapsto \mu\rho.U]U_0 \longrightarrow U'$. If $n = 0$ (i.e., $\mu\rho.U$ is never expanded before the reduction), then there exists U'_0 such that $[\rho' \mapsto \mu\rho.U]U'_0 \preceq U'$ (which also implies $[\rho' \mapsto \mu\rho.U]U'_0 \leq U'$ by the first law of Lemma 11.) and $U_0 \longrightarrow U'_0$. Therefore, $U'' = [\rho' \mapsto U_1]U'_0$ satisfies the required conditions. If $n = k+1$, then there exists U'_0 such that $[\rho' \mapsto \mu\rho.U]U_0 \preceq U'_0 \longrightarrow U'$, where the expansion rule is used k times before the reduction in $U'_0 \longrightarrow U'$ and U'_0 is obtained from by replacing one occurrence of $\mu\rho.U$ in $[\rho' \mapsto \mu\rho.U]U_0$ with $[\rho \mapsto \mu\rho.U]U$. Let U''_0 be the usage obtained from $[\rho' \mapsto U_1]U_0$ by replacing the corresponding occurrence of U_1 with $[\rho \mapsto U_1]U$. By the induction hypothesis, there exists U''' such that $U''_0 \longrightarrow U'''$ and $U''' \leq \mathcal{R} \leq U'$. Moreover, by the condition $U_1 \leq [\rho \mapsto U_1]U$, we have $[\rho' \mapsto U_1]U_0 \leq U''_0$. So, by the condition $[\rho' \mapsto U_1]U_0 \leq U''_0$ and $U''_0 \longrightarrow U'''$, there exists U'' such that $[\rho' \mapsto U_1]U_0 \longrightarrow U''$ and $U'' \leq U'''$. Therefore, we obtain $U'' \leq U''' \leq \mathcal{R} \leq U'$ as required.

The fourth condition follows from the fact that $ob_\alpha(\mu\rho.U)$ is the least fixed-point for $\lambda x.ob^{\rho \mapsto x}_\alpha(U)$ and $ob_\alpha(U_1) \geq ob^{\{\rho \mapsto ob_\alpha(U_1)\}}_\alpha(U)$, which implies $ob_\alpha(U_1) \geq ob_\alpha(\mu\rho.U)$. (This is the very reason why we defined the obligation level of a recursive usage as the least fixed-point.)

Similar observation for the capability level yields the third condition $cap_\alpha(U_1) \leq cap_\alpha(\mu\rho.U)$. □

Lemma 13 *If* $\Gamma \leq \Gamma'$ *and* $\Gamma' \longrightarrow \Delta'$, *then there exists* Δ *such that* $\Gamma \longrightarrow \Delta$ *and* $\Delta \leq \Delta'$.

Proof By the definition of the subusage relation, if $U \leq U'$ and $U' \longrightarrow V'$, then there exists V such that $U \longrightarrow V$ and $V \leq V'$. So, the lemma follows immediately. □

Lemma 14 *If $rel(U)$ and $U \longrightarrow U'$, then $rel(U')$. If $rel(U)$ and $U \leq U'$, then $rel(U')$.*

Proof The first property follows immediately from the definition of rel. The second one follows immediately from Lemma 2. □

As a corollary of the above lemma, we obtain the following property.

Lemma 15 *If $rel(\Gamma)$ and $\Gamma \longrightarrow^* \leq \Delta$, then $rel(\Delta)$.*

Proof This follows immediately from Lemma 14. □

Lemma 16 *If $\Gamma, x:\tau \vdash_l P$ and $x \notin FV(P)$, then $ob(\tau) = \infty$ and $\Gamma \vdash_l P$.*

Proof Straightforward induction on derivation of $\Gamma, x:\tau \vdash_l P$. □

Lemma 17 *If $\Gamma \vdash_l P$ and $P \preceq Q$, then $\Gamma \vdash_l Q$.*

Proof The proof proceeds by induction on derivation of $P \preceq Q$. We show only main cases. The other cases are trivial.

- Case for S-NEW: There are two cases to consider.
  - Case where $P = (\nu x:\xi)\, P_1 \mid P_2$ and $Q = (\nu x:\xi)\, (P_1 \mid P_2)$: Suppose $\Gamma \vdash_l P$. Then there exist Γ_1, Γ_2, and U such that:
    $\Gamma_1, x:\xi/U \vdash_l P_1$
    $\Gamma_2 \vdash_l P_2$
    $\Gamma \leq \Gamma_1 \mid \Gamma_2$
    $rel(U)$
    We can assume without loss of generality that $x \notin dom(\Gamma_2)$. So, we have $(\Gamma_1 \mid \Gamma_2), x:\xi/U \vdash_l P_1 \mid P_2$, from which $\Gamma \vdash_l Q$ follows.
  - Case where $P = (\nu x:\xi)\, (P_1 \mid P_2)$ and $Q = (\nu x:\xi)\, P_1 \mid P_2$: Suppose $\Gamma \vdash_l P$. Then there exist Γ_1, Γ_2, and U such that:
    $\Gamma_1 \vdash_l P_1$
    $\Gamma_2 \vdash_l P_2$
    $\Gamma \leq (\Gamma_1 \mid \Gamma_2)\backslash\{x\}$
    $rel((\Gamma_1 \mid \Gamma_2)(x))$
    If $x \notin dom(\Gamma_2)$, then the result follows immediately. If $\Gamma_2 = \xi/U_2$, then by Lemma 16, $\Gamma_2\backslash\{x\} \vdash_l P_2$ and $ob(U_2) = \infty$. So, by Lemma 11, we can apply T-SUB to $\Gamma_1 \vdash_l P_1$ and obtain $\Gamma_1 \mid x:\xi/U_2 \vdash_l P_1$. By using T-NEW, T-PAR, and T-SUB, we obtain $\Gamma \vdash_l Q$ as required.
- Case for S-IFT: In this case, $P = \mathbf{if}\ \mathit{true}^{l'}\ \mathbf{then}\ Q\ \mathbf{else}\ Q'$. $\Gamma \vdash_l Q$ follows immediately from the typing rules.
- Case for S-REP: In this case, $P = *P_1$ and $Q = *P_1 \mid P_1$. If $\Gamma \vdash_l P$, then there exists Γ_1 such that $\Gamma_1 \vdash_l P_1$ and $\Gamma \leq *\Gamma_1$. Since $\Gamma \leq *\Gamma_1 \leq *\Gamma_1 \mid \Gamma_1$, we obtain $\Gamma \vdash_l Q$ by using T-REP, T-PAR, and T-SUB.

Lemma 18 *If $\Gamma \leq \Delta$ and $[x \mapsto v]\Gamma$ is well defined, then $[x \mapsto v]\Delta$ is also well defined and $[x \mapsto v]\Gamma \leq [x \mapsto v]\Delta$ holds.*

Proof $[x \mapsto v]\Delta$ is well defined since for every $y \in dom(\Delta)$, $\Delta(y) \sim \Gamma(y)$ holds. $[x \mapsto v]\Gamma \leq [x \mapsto v]\Delta$ follows from the fact that $U_1 \leq U_1'$ and $U_2 \leq U_2'$ imply $U_1 \mid U_2 \leq U_1' \mid U_2'$.

Lemma 19 (substitution lemma) *If $\Gamma, x:\tau \vdash_l P$ and $[x \mapsto v]\Gamma$ is well defined, then $[x \mapsto v]\Gamma \vdash_l [x \mapsto v]P$ holds.*

Proof This follows by induction on the structure of P. We show only the case where P is an output process. The other cases are similar or trivial.

Suppose $P = \overline{y}\langle\widetilde{w}\rangle\, P_1$. Then, the following conditions must hold:

$$\begin{array}{l}\Gamma_1, y : \langle\widetilde{\sigma}\rangle^{l_1}/U \vdash_{l_2} P_1\\ l \sqsubseteq l_1, l_2\\ \widetilde{\sigma'} \leq \uparrow\widetilde{\sigma}\\ t_c = \infty \Rightarrow l_1 \sqsubseteq l_2\\ \Gamma \leq \uparrow^{(t_c+1,t_c+1)}(\Gamma_1 \mid \widetilde{w} : \widetilde{\sigma'}) \mid y : \langle\widetilde{\sigma}\rangle^{l_1}/O^0_{t_c}.U\end{array}$$

From the last condition and Lemma 18, it follows that $[x \mapsto v]\Gamma_1 \mid [x \mapsto v]y : \langle\widetilde{\sigma}\rangle^{l_1}/U$ is well defined. So, by the induction hypothesis, we have $[x \mapsto v]\Gamma_1 \mid [x \mapsto v]y : \langle\widetilde{\sigma}\rangle^{l_1}/U \vdash_{l_2} [x \mapsto v]P_1$.

We perform case analysis on $[x \mapsto v]y$. If $[x \mapsto v]y = v$ (i.e., $y = x$ or $y = v$), then v must be a variable. We can assume without loss of generality that $[x \mapsto v]\Gamma_1 = \Gamma_1', v : \langle\widetilde{\sigma}\rangle^{l_1}/U'$ (since we can add the binding $x : \langle\widetilde{\sigma}\rangle^{l_1}/\mathbf{0}$ or $v : \langle\widetilde{\sigma}\rangle^{l_1}/\mathbf{0}$ if $v \notin [x \mapsto v]\Gamma_1$). By using T-OUT, we obtain:

$$\uparrow^{(t_c+1,t_c+1)}(\Gamma_1' \mid \widetilde{w}' : \widetilde{\sigma'}) \mid v : \langle\widetilde{\sigma}\rangle^{l_1}/O^0_{t_c}.(U' \mid U) \vdash_{l_2} [x \mapsto v]P$$

where $\widetilde{w}' = [x \mapsto v]\widetilde{w}$. We obtain $[x \mapsto v]\Gamma \vdash_l [x \mapsto v]P$ by using T-SUB, since

$$\begin{array}{rl}[x \mapsto v]\Gamma & \leq [x \mapsto v](\uparrow^{(t_c+1,t_c+1)}(\Gamma_1 \mid \widetilde{w} : \widetilde{\sigma'}) \mid y : \langle\widetilde{\sigma}\rangle^{l_1}/O^0_{t_c}.U)\\ & \leq \uparrow^{(t_c+1,t_c+1)}(\Gamma_1' \mid \widetilde{w'} : \widetilde{\sigma'}) \mid v : \langle\widetilde{\sigma}\rangle^{l_1}/(\uparrow^{(t_c+1,t_c+1)}U' \mid O^0_{t_c}.U)\\ & \leq \uparrow^{(t_c+1,t_c+1)}(\Gamma_1' \mid \widetilde{w'} : \widetilde{\sigma'}) \mid v : \langle\widetilde{\sigma}\rangle^{l_1}/O^0_{t_c}.(U' \mid U)\end{array}$$

The last relation is obtained by $\uparrow^{(t_c+1,t_c+1)}U' \mid O^0_{t_c}.U \leq O^0_{t_c}.(\uparrow^{(t_c+1,t_c+1)}U' \mid U) \leq O^0_{t_c}.(U' \mid U)$, using Lemma 11.

If $[x \mapsto v]y \neq v$ (i.e., $y \neq x$ and $y \neq v$, which also imply $[x \mapsto v]y = y$), we have $[x \mapsto v]\Gamma_1, y : \langle\widetilde{\sigma}\rangle^{l_1}/U \vdash_{l_2} [x \mapsto v]P_1$. By applying T-OUT, we obtain $\uparrow^{(t_c+1,t_c+1)}([x \mapsto v]\Gamma_1 \mid \widetilde{w}' : \widetilde{\sigma'}) \mid y : \langle\widetilde{\sigma}\rangle^{l_1}/O^0_{t_c}.U \vdash_{l_2} [x \mapsto v]P$ where $\widetilde{w}' = [x \mapsto v]\widetilde{w}$. By using T-SUB, we get $[x \mapsto v]\Gamma \vdash_l [x \mapsto v]P$ as required.

Proof of Theorem 1 The proof proceeds by induction on derivation of $P \longrightarrow Q$, with case analysis on the last rule used.

- Case for R-COM: In this case, $P = \overline{x}\langle\widetilde{v}\rangle.\, P_1 \mid x(\widetilde{y}).\, P_2$ and $Q = P_1 \mid [\widetilde{y} \mapsto \widetilde{v}]P_2$. By the typing rules, it must be the case that:

  $$\begin{array}{l}\Gamma_1, x : \langle\widetilde{\tau}\rangle^{l_1}/U_1 \vdash_l P_1\\ \Gamma_2, x : \langle\widetilde{\tau}\rangle^{l_2}/U_2, \widetilde{y} : \widetilde{\tau} \vdash_l P_2\\ \widetilde{\tau'} \leq \uparrow\widetilde{\tau}\\ \Gamma \leq (\uparrow^{(t_1+1,t_1+1)}(\Gamma_1 \mid \widetilde{v} : \widetilde{\tau'}) \mid x : \langle\widetilde{\tau}\rangle^{l_1}/O^0_{t_1}.U_1)\\ \qquad \mid (\uparrow^{(t_2+1,t_2+1)}\Gamma_2, x : \langle\widetilde{\tau}\rangle^{l_2}/I^0_{t_2}.U_2)\\ l \sqsubseteq l_1, l_2\end{array}$$

  By the substitution lemma (Lemma 19), we have:

  $$(\Gamma_2, x : \langle\widetilde{\tau}\rangle^{l_2}/U_2) \mid \widetilde{v} : \widetilde{\tau} \vdash_l [\widetilde{y} \mapsto \widetilde{v}]P_2.$$

  So, $\Delta' \vdash_l Q$ holds for $\Delta' = (\Gamma_1 \mid \widetilde{v} : \widetilde{\tau} \mid x : \langle\widetilde{\tau}\rangle^{l_1}/U_1) \mid (\Gamma_2, x : \langle\widetilde{\tau}\rangle^{l_2}/U_2)$. Moreover, we obtain $\Gamma \leq (\Gamma_1 \mid \widetilde{v} : \widetilde{\tau} \mid x : \langle\widetilde{\tau}\rangle^{l_1}/O^0_{t_1}.U_1) \mid (\Gamma_2, x : \langle\widetilde{\tau}\rangle^{l_2}/I^0_{t_2}.U_2) \longrightarrow \Delta'$ by using Lemma 11. By Lemma 13, there exists Δ such that $\Gamma \longrightarrow \Delta$ and $\Delta \leq \Delta'$. So, we have $\Delta \vdash_l Q$ and $\Gamma \longrightarrow \Delta$ as required.
- Case for R-PAR: In this case, $P = P_1 \mid P_2$ and $Q = Q_1 \mid P_2$ with $P_1 \longrightarrow Q_1$. By the typing rules, there exist Γ_1 and Γ_2 such that $\Gamma_i \vdash_l P_i$ and $\Gamma \leq \Gamma_1 \mid \Gamma_2$. By the induction hypothesis, there exists Δ_1 such that $\Gamma_1 \longrightarrow \Delta_1$ or $\Delta_1 = \Gamma_1$. Let $\Delta' = \Delta_1 \mid \Gamma_2$. Then, $\Delta' \vdash_l Q$, and either $\Gamma_1 \mid \Gamma_2 \longrightarrow \Delta'$ or $\Delta' = \Gamma_1 \mid \Gamma_2$ holds. In the latter case, the required result holds for $\Delta = \Gamma$. In the former case, by Lemma 13, there exists Δ such that $\Gamma \longrightarrow \Delta$ and $\Delta \leq \Delta'$. By using T-SUB, we obtain $\Delta \vdash_l Q$ as required.

– Case for R-NEW: In this case, $P = (\nu x:\xi)\, P_1$ and $Q = (\nu x:\xi)\, Q_1$ with $P_1 \longrightarrow Q_1$. By the typing rules, $\Gamma, x:\xi/U \vdash_l P_1$ with $rel(U)$. By the induction hypothesis, either $\Gamma, x:\xi/U \vdash_l Q_1$ holds, or there exists Δ and U' such that $\Delta, x:\xi/U' \vdash_l Q_1$ and $\Gamma, x:\xi/U \longrightarrow \Delta, x:\xi/U'$. In the former case, $\Delta = \Gamma$ satisfies the required condition. In the latter case, by Lemma 14, $rel(U')$ holds, so that we can obtain $\Delta \vdash_l Q$ as required.
– Case for R-SP: This follows immediately from Lemma 17 and the induction hypothesis.

□

B Proof of Theorem 4

This section gives a proof of Theorem 4, which states that the erasure function preserves barbed bisimilarity. We first introduce miscellaneous definitions needed for proving theorems in Section B.1, and prove some basic properties about the reduction relation and the erasure function in Section B.2. We then show, in Section B.3, a key theorem about the lock-freedom (Theorem 5), which states that any communication with a finite-level capability will eventually succeed. Using the theorem, we show that a process P and its erasure $\mathbf{Er}_\Gamma(P)$ simulate each other (Sections B.4 and B.5), which proves Theorem 4 (Section B.6).

B.1 Miscellaneous definitions

We first define the relation $\sim$ on types, Intuitively, $\tau \sim \tau'$ holds when τ and τ' are identical except for their outermost usages.

Definition 26 The relation $\tau \sim \tau'$ is the least equivalence relation satisfying the rules: $\mathbf{unit} \sim \mathbf{unit}, \mathbf{bool}^l \sim \mathbf{bool}^l$ and $\langle \tau_1, \ldots, \tau_n \rangle^l / U \sim \langle \tau_1, \ldots, \tau_n \rangle^l / U'$ for $l \in \{\mathbf{H}, \mathbf{L}\}$. The relation $\sim$ is extended to a relation on type environments by: $\Gamma \sim \Gamma'$ if and only if $dom(\Gamma) = dom(\Gamma')$ and $\Gamma(x) \sim \Gamma'(x)$ for every $x \in dom(\Gamma)$.

Definition 27 $ob_\alpha(\tau)$ is defined by:

$$ob_\alpha(\mathbf{unit}) = ob_\alpha(\mathbf{bool}^l) = \infty$$
$$ob_\alpha(\xi/U) = ob_\alpha(U)$$

Definition 28 (size of process) The size of a process P, written $\#(P)$, is defined by:

$$\#(\mathbf{0}) = 0$$
$$\#(\overline{x}\langle\widetilde{v}\rangle.\, P) = \#(x(\widetilde{y}).\, P) = \#(P) + 1$$
$$\#(*P) = \#((\nu x:\xi)\, P) = \#(P) + 1$$
$$\#(P \mid Q) = \#(\mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q) = \#(P) + \#(Q) + 1$$

Definition 29 (strong barbs) The *strong barbs* of P, written $\mathit{SBarbs}(P)$, is defined by:

$$\mathit{SBarbs}(P) = \{\overline{x} \mid P \preceq (\nu\widetilde{y})\,(\overline{x}\langle\widetilde{v}\rangle.\, Q \mid R), x \notin \{\widetilde{y}\}\}$$
$$\cup \{x \mid P \preceq (\nu\widetilde{y})\,(x(\widetilde{z}).\, Q \mid R), x \notin \{\widetilde{y}\}\}$$

For the sake of technical convenience, we sometimes assume that each input/output process is annotated with its security level and capability level:

$$P ::= \overline{x}\langle v_1, \ldots, v_n\rangle^{l,t}.\, P \mid x(y_1, \ldots, y_n)^{l,t}.\, P \mid \cdots$$

The typing rules for annotated input/output processes are given as follows.

$$\frac{\Gamma, x:\langle\widetilde{\tau}\rangle^{l_1}/U \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\uparrow^{(t_c+1, t_c+1)}(\Gamma \mid \widetilde{v}:\uparrow\widetilde{\tau}) \mid x:\langle\widetilde{\tau}\rangle^{l_1}/O^0_{t_c}.U \vdash_l \overline{x}\langle\widetilde{v}\rangle^{l_1,t_c}.\, P} \qquad \text{(T-OUT')}$$

$$\frac{\Gamma, x:\langle\widetilde{\tau}\rangle^{l_1}/U, \widetilde{y}:\widetilde{\tau} \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\uparrow^{(t_c+1,t_c+1)}\Gamma, x:\langle\widetilde{\tau}\rangle^{l_1}/I^0_{t_c}.U \vdash_l x(\widetilde{y})^{l_1,t_c}.\, P} \quad \text{(T-IN')}$$

The only change from T-OUT and T-IN is that the processes are annotated with a security level and a capability level. Given a type derivation tree of an unannotated process term, we can always recover annotations based on the above rules.

We also annotate the reduction relation with a security level. Intuitively, $P \longrightarrow_l Q$ means that P is reduced to Q by communication on a channel whose security level is l.

Definition 30 The relation $\longrightarrow_l$ is defined by:

$$\overline{x}\langle\widetilde{v}\rangle^{l,t_1}.\, P \mid x(\widetilde{y})^{l,t_2}.\, Q \longrightarrow_l P \mid [\widetilde{y} \mapsto \widetilde{v}]Q \quad \text{(R-COM)}$$

$$\frac{P \longrightarrow_l Q}{P \mid R \longrightarrow_l Q \mid R} \quad \text{(R-PAR)}$$

$$\frac{P \longrightarrow_l Q}{(\nu x:\xi)\, P \longrightarrow_l (\nu x:\xi)\, Q} \quad \text{(R-NEW)}$$

$$\frac{P \preceq P' \qquad P' \longrightarrow_l Q' \qquad Q' \preceq Q}{P \longrightarrow_l Q} \quad \text{(R-SPCONG)}$$

Note that if a process P is well-typed, $P \longrightarrow Q$ holds if and only if $P \longrightarrow_l Q$ holds for some l.

We refine the definition of contexts, so that a hole is annotated with a substitution and a set of variables. The substitution is applied when a process is put into the hole.

Definition 31 (extended contexts) The set of *extended contexts* is given by:

$$\begin{array}{rl} C ::= & [\,]_{\theta,S} \mid \overline{x}\langle\widetilde{v}\rangle^{l,t}.\, C \mid x(\widetilde{y})^{l,t}.\, C \\ & \mid (P \mid C) \mid (C \mid P) \mid *C \mid (\nu x:\xi)\, C \\ & \mid \mathbf{if}\ v\ \mathbf{then}\ C\ \mathbf{else}\ P \mid \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ C \end{array}$$

Here, θ ranges over the set of substitutions, and S ranges over the powerset of variables. If the hole in the context C is $[\,]_{\theta,S}$ and $FV(P) \subseteq S$, $C[P]$ is the process obtained by replacing $[\,]_{\theta,S}$ with θP.

We extend the relations $\preceq$ and $\longrightarrow_l$ on processes to relations on contexts by defining $FV([\,]_{\theta,S})$ to be $\theta S = \{\theta x \mid x \in S\} \cap \mathbf{Var}$, and defining the substitution for the hole by: $\theta_1[\,]_{\theta_2,S} = [\,]_{\theta_1\circ\theta_2,S}$ (where $\theta_1 \circ \theta_2$ is the composition of substitutions). For example, $\overline{x}\langle v\rangle.\, P \mid x(y).\, [\,]_{id,\{y\}} \longrightarrow P \mid [\,]_{[y\mapsto v]\{y\}}$ where *id* is the identity substitution. Note that if $C \longrightarrow_l C'$ and $C[P]$ is well defined, then $C[P] \longrightarrow_l C'[P]$ holds. We also define $\mathbf{Er}_\Gamma(C)$ by adding the clause $\mathbf{Er}_\Gamma([\,]_{\theta,S}) = [\,]_{\mathbf{Er}_\Gamma(\theta),S}$ where $\mathbf{Er}_\Gamma([x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]) = [x_1 \mapsto \mathbf{Er}_\Gamma(v_1), \ldots, x_n \mapsto \mathbf{Er}_\Gamma(v_n)]$.

We introduce two subsets of (extended) contexts: the set of finite-level contexts and the set of evaluation contexts. A *finite-level context* is a context whose hole is guarded only by input/output prefixes with finite capability levels. An *evaluation context* is a context whose hole is not guarded by any input/output prefixes.

Definition 32 (finite-level context) The set of *finite-level contexts* is given by:

$$\begin{array}{rl} C ::= & [\,]_{\theta,S} \mid \overline{x}\langle\widetilde{v}\rangle^{\mathbf{H},n}.\, C \mid x(\widetilde{y})^{\mathbf{H},n}.\, C \\ & \mid (P \mid C) \mid (C \mid P) \mid (\nu x:\xi)\, C \end{array}$$

Here, n ranges over $\mathbf{Nat}$.

Definition 33 (depth of context, evaluation context) Let C be a finite level context. The *depth* of C, written *depth*(C), is defined by:

$$\begin{array}{l}\mathit{depth}([\,]_{\theta,S}) = 0\\ \mathit{depth}(\overline{x}\langle\widetilde{v}\rangle^n.\, C) = \mathit{depth}(C) + 1\\ \mathit{depth}(x(\widetilde{y})^n.\, C) = \mathit{depth}(C) + 1\\ \mathit{depth}(P \mid C) = \mathit{depth}(C \mid P) = \mathit{depth}((\nu x : \xi)\, C) = \mathit{depth}(C)\end{array}$$

A finite level context whose depth is 0 is called an *evaluation context*.

A *context with two holes* is a term obtained from a process by replacing one sub-process with $[\,]^{(1)}_{\theta_1,S_1}$ and another sub-process with $[\,]^{(2)}_{\theta_2,S_2}$. If C is a context with two holes, we write $C[P_1, P_2]$ for the process obtained from C by replacing $[\,]^{(1)}_{\theta_1,S_1}$ and $[\,]^{(2)}_{\theta_2,S_2}$ with $\theta_1 P_1$ and $\theta_2 P_2$ respectively. *Finite-level contexts with two holes* and *evaluation contexts with two holes* are defined in a similar manner.

Definition 34 Let C be an extended context. $ext(\Gamma, C)$ is the type environment defined by:

$$\begin{array}{l}ext(\Gamma, [\,]_{\theta_S}) = \Gamma\\ ext(\Gamma, \overline{x}\langle\widetilde{v}\rangle^t.\, C) = ext(\Gamma, C)\\ ext(\Gamma, x(\widetilde{y})^t.\, C) = ext((\Gamma, \widetilde{y} : \widetilde{\tau}), C) \text{ (if } \Gamma(x) = \langle\widetilde{\tau}\rangle^l/U)\\ ext(\Gamma, P \mid C) = ext(\Gamma, C \mid P) = ext(\Gamma, *C) = ext(\Gamma, C)\\ ext(\Gamma, (\nu x : \xi)\, C) = ext((\Gamma, x : \xi/\mathbf{0}), C)\\ ext(\Gamma, \mathbf{if}\ v\ \mathbf{then}\ C\ \mathbf{else}\ P) = ext(\Gamma, C)\\ ext(\Gamma, \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ C) = ext(\Gamma, C)\end{array}$$

Intuitively, $ext(\Gamma, C)$ is the type environment obtained by adding to Γ bindings on the variables bound by C. Note that if the hole in C is of the form $[\,]_{id,S}$, then $\mathbf{Er}_\Gamma(C[P]) = \mathbf{Er}_\Gamma(C)[\mathbf{Er}_\Delta(P)]$ where $\Delta = ext(\Gamma, C)$. If C is a context with two holes, we write $ext^{(1)}(\Gamma, C)$ and $ext^{(2)}(\Gamma, C)$ for $ext(\Gamma, C[[\,]_{\theta,S}, \mathbf{0}])$ and $ext(\Gamma, C[\mathbf{0}, [\,]_{\theta,S}])$ respectively.

B.2 Basic Properties

Lemma 20 *If* $P \longrightarrow Q$, *then* $P \preceq (\nu\widetilde{u})\,(\overline{x}\langle\widetilde{v}\rangle.\, P_1 \mid x(\widetilde{y}).\, P_2 \mid P_3)$ *and* $(\nu\widetilde{u})\,(P_1 \mid [\widetilde{y} \mapsto \widetilde{v}]P_2 \mid P_3) \preceq Q$ *for some* $\widetilde{u}, x, \widetilde{v}, P_1, P_2, P_3$.

Proof This follows by straightforward induction on derivation of $P \longrightarrow Q$. □

Lemma 21 *If* $\Gamma \sim \Delta$, *then* $\mathbf{Er}_\Gamma(P) = \mathbf{Er}_\Delta(P)$.

Proof Straightforward induction on the structure of P. □

Lemma 22 *If* $\Gamma(v) \sim \tau$,
then $\boldsymbol{ErV}_\Gamma([x \mapsto v]v') = [x \mapsto \boldsymbol{ErV}_\Gamma(v)]\boldsymbol{ErV}_{\Gamma, x:\tau}(v')$.

Proof If $x \neq v'$, then $\mathbf{ErV}_\Gamma([x \mapsto v]v') = \mathbf{ErV}_\Gamma(v') = \mathbf{ErV}_{\Gamma,x:\tau}(v') = [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{ErV}_{\Gamma,x:\tau}(v')$. Suppose that $x = v'$. If $\mathbf{High}(\tau)$, then the result follows from $\mathbf{ErV}_\Gamma(v) = \mathbf{ErV}_{\Gamma,x:\tau}(x) = \star$. Otherwise, $\mathbf{ErV}_{\Gamma,x:\tau}(x) = x$. So, we have $\mathbf{ErV}_\Gamma([x \mapsto v]v') = \mathbf{ErV}_\Gamma(v) = [x \mapsto \mathbf{ErV}_\Gamma(v)]x = [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{ErV}_{\Gamma,x:\tau}(x)$. □

Lemma 23 *If* $\Gamma(v) \sim \tau$,
then $\mathbf{Er}_\Gamma([x \mapsto v]P) = [x \mapsto \boldsymbol{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(P)$.

Proof This follows by straightforward induction on the structure of P. We show only the case for output processes. The other cases are similar or trivial.

– Case where P is of the form $\overline{y}\langle\widetilde{w}\rangle.\,Q$.
By the induction hypothesis, $\mathbf{Er}_\Gamma([x \mapsto v]Q) = [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(Q)$. Note that $\Gamma([x \mapsto v]y) \sim \langle\widetilde{\sigma}\rangle^{\mathbf{L}}/\mathbf{0}$ if and only if $(\Gamma, x:\tau)(y) \sim \langle\widetilde{\sigma}\rangle^{\mathbf{L}}/\mathbf{0}$. So, if $\Gamma([x \mapsto v]y) \not\sim \langle\widetilde{\sigma}\rangle^{\mathbf{L}}/\mathbf{0}$, then the result follows from the following equations.

$$\begin{aligned}&\mathbf{Er}_\Gamma([x \mapsto v]P)\\ =\ &\mathbf{Er}_\Gamma([x \mapsto v]Q)\\ =\ &[x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(Q)\\ =\ &[x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(P)\end{aligned}$$

Suppose that $\Gamma([x \mapsto v]y) \sim \langle\widetilde{\sigma}\rangle^{\mathbf{L}}/\mathbf{0}$. Let $y' = [x \mapsto v]y$. Then, $y' = \mathbf{ErV}_\Gamma(y') = [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{ErV}_{\Gamma,x:\tau}(y) = [x \mapsto \mathbf{ErV}_\Gamma(v)]y$. (The second equality follows from Lemma 22.) Thus, the result follows from the following equations.

$$\begin{aligned}&\mathbf{Er}_\Gamma([x \mapsto v]P)\\ =\ &\overline{y'}\langle\mathbf{ErV}_\Gamma([x \mapsto v]\widetilde{w})\rangle.\,\mathbf{Er}_\Gamma([x \mapsto v]Q)\\ =\ &\overline{y'}\langle[x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{ErV}_{\Gamma,x:\tau}(\widetilde{w})\rangle.\\ &\qquad[x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(Q)\\ =\ &[x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(P)\end{aligned}$$

□

Lemma 24 *If* $\mathbf{High}(\Gamma(x))$*, then* $x \notin \mathbf{Er}_\Gamma(P)$.

Proof Straightforward induction on the structure of P. □

B.3 Lock-freedom Property

In this subsection, we show that input/output actions whose capability levels are finite succeed eventually.

We define x^α by: $x^I = x$ and $x^O = \overline{x}$.

Lemma 25 *Suppose that the following conditions hold.*

1. $\Gamma \vdash_l P$
2. $\Delta \vdash_l Q$
3. $ob_\alpha(\Gamma(x)) \in \mathbf{Nat}$
4. $rel(\Gamma \mid \Delta)$

Then there exists R *such that* $P \mid Q \longrightarrow^*_{\mathbf{H}} R$ *and* $x^\alpha \in \mathit{SBarbs}(R)$.

Proof Let n be $ob_\alpha(\Gamma(x))$. The proof proceeds by well-founded induction on $(n, \#(P))$, where the well-founded order is defined by $(n, m) < (n', m') \iff (n < n') \vee (n = n' \wedge m < m')$. Let U_x be the usage of type $\Gamma(x)$. Without loss of generality we assume below that $\mathit{con}_{\overline{\alpha}}(U_x)$ holds; otherwise $\mathit{cap}_{\overline{\alpha}}(U_x) < n$, from which we obtain $ob_\alpha(\Delta(x)) < n$ using the condition $rel(\Gamma \mid \Delta)$. So, we can switch the role of P and Q and use induction hypothesis to conclude $P \mid Q \longrightarrow^*_{\mathbf{H}} R$ and $x^\alpha \in \mathit{SBarbs}(R)$ for some R.

We perform case analysis on P. We will consider only the case for $\alpha = O$ below: The case for $\alpha = I$ is similar.

– Case $P = \mathbf{0}$: This case cannot happen.
– Case $P = \overline{y}\langle\widetilde{v}\rangle.\,P_1$. If $y = x$, then the result follows immediately. If $y \neq x$, then by the rule T-OUT and T-WEAK, it must be the case that:

$$\begin{aligned}&\Gamma_1, y:\langle\widetilde{\tau}\rangle^{l_1}/U \vdash_{l'} P_1\\ &\widetilde{\tau}' \leq \uparrow\widetilde{\tau}\\ &\Gamma \leq \uparrow^{(t+1,t+1)}(\Gamma_1 \mid \widetilde{v}:\widetilde{\tau}'), y:\langle\widetilde{\tau}\rangle^{l_1}/O^0_t.U\end{aligned}$$

The third condition implies $t < ob_O(\uparrow^{(t+1,t+1)}(\Gamma_1 \mid \widetilde{v}:\widetilde{\tau}')(x)) \leq ob_O(\Gamma(x)) = n$ (which also implies $l_1 = \mathbf{H}$ by the well-formedness condition of types). Since $rel(\Gamma \mid \Delta)$ holds, Lemma 15 implies $rel((\uparrow^{(t+1,t+1)}(\Gamma_1 \mid \widetilde{v}:\widetilde{\tau}'), y:\langle\widetilde{\tau}\rangle^{l_1}/O^0_t.U) \mid \Delta)$, from which we obtain $rel(\langle\widetilde{\tau}\rangle^{l_1}/O^0_t.U \mid \Delta(y))$. So, it must be the case that $ob_I(\Delta)(y) \leq t < n$.

By the induction hypothesis, it must be the case that $Q \longrightarrow^*_{\mathbf{H}} \preceq (\nu \widetilde{u})\,(y(\widetilde{z}).\,Q_1 \mid Q_2)$. By Lemma 17 and Theorem 1, $\Delta' \vdash_l (\nu \widetilde{u})\,(y(\widetilde{z}).\,Q_1 \mid Q_2)$ holds for some Δ' such that $\Delta \longrightarrow^* \Delta'$. By the typing rules, it must be the case that

$\Delta_1, y : \langle \widetilde{\tau} \rangle^{\mathbf{H}}/V, \widetilde{z} : \widetilde{\tau} \vdash_{l_1} Q_1$
$\Delta_2 \vdash_l Q_2$
$\Delta', \widetilde{u} : \widetilde{\sigma} \leq (\uparrow^{(t_2+1, t_2+1)} \Delta_1, y : \langle \widetilde{\tau} \rangle^{\mathbf{H}}/I^{t_1}_{t_2}.V) \mid \Delta_2$

By the substitution lemma (Lemma 19), $(\Delta_1, y : \langle \widetilde{\tau} \rangle^{\mathbf{H}}/V) \mid \widetilde{v} : \widetilde{\tau} \vdash_{l_1} [\widetilde{z} \mapsto \widetilde{v}]Q_1$. So, we have:

$(\Gamma_1 \mid \widetilde{v} : \widetilde{\tau}, y : \langle \widetilde{\tau} \rangle^{\mathbf{H}}/U \mid V) \mid \Delta_1 \mid \Delta_2 \vdash_l P_1 \mid [\widetilde{z} \mapsto \widetilde{v}]Q_1 \mid Q_2$

Moreover, $rel(\Gamma \mid \Delta)$ and Lemma 14 imply $rel((\Gamma_1, y : \langle \widetilde{\tau} \rangle^{\mathbf{H}}/(U \mid V)) \mid \Delta_1 \mid \Delta_2)$. The condition $\Gamma \leq \uparrow^{(t+1,t+1)}(\Gamma_1 \mid \widetilde{v} : \widetilde{\tau}'), y : \langle \widetilde{\tau} \rangle^{l_1}/O^0_t.U$ implies $ob_O(\Gamma_1 \mid \widetilde{v} : \widetilde{\tau}')(x) \leq ob_O \uparrow^{(t+1,t+1)}(\Gamma_1 \mid \widetilde{v} : \widetilde{\tau}')(x) \leq ob_O(\Gamma(x)) = n$, which implies either (i)$ob_O(\Gamma_1(x)) \leq n$ or (ii)$ob_O((\widetilde{v} : \widetilde{\tau}')(x)) \leq n$. In the former case, since $\#P_1 < \#P$ holds, the induction hypothesis implies that there must exist R such that $P_1 \mid [\widetilde{z} \mapsto \widetilde{v}]Q_1 \mid Q_2 \longrightarrow^*_{\mathbf{H}} R$ and $\overline{x} \in \mathit{SBarbs}(R)$. In the latter case, $ob_O((\widetilde{v} : \widetilde{\tau})(x)) < ob_O((\widetilde{v} : \widetilde{\tau}')(x)) \leq n$. So, by the induction hypothesis, $P_1 \mid [\widetilde{z} \mapsto \widetilde{v}]Q_1 \mid Q_2 \longrightarrow_{\mathbf{H}} * R$ with $\overline{x} \in \mathit{SBarbs}(R)$. The required result follows, since $P \mid Q \longrightarrow^*_{\mathbf{H}} (\nu \widetilde{u})\,(P_1 \mid [\widetilde{z} \mapsto \widetilde{v}]Q_1 \mid Q_2)$.

- Case for $P = y(\widetilde{z}).\,P_1$: Similar to the above case.
- Case for $P = P_1 \mid P_2$: There exist Γ_1 and Γ_2 such that $\Gamma \leq \Gamma_1 \mid \Gamma_2$ and $\Gamma_i \vdash_l P_i$. By Lemma 15, $rel(\Gamma_1 \mid \Gamma_2 \mid \Delta)$ holds. Since $ob_O(\Gamma(x)) = n$, either $ob_O(\Gamma_1(x)) \leq n$ or $ob_O(\Gamma_2(x)) \leq n$ holds. In the former case, since $\#(P_1) < \#(P)$ holds, we can apply induction hypothesis to obtain $P \mid Q \preceq P_1 \mid (P_2 \mid Q) \longrightarrow_{\mathbf{H}} * R$ with $\overline{x} \in \mathit{SBarbs}(R)$. The latter case is similar.
- Case for $P = *P_1$: There exists Γ_1 such that $\Gamma_1 \vdash_l P_1$ and $\Gamma \leq *\Gamma_1$. Since $\Gamma \leq \Gamma_1 \mid \Gamma$, $rel(\Gamma_1 \mid (\Gamma \mid \Delta))$ holds. So, by applying the induction hypothesis to P_1, we obtain R such that $P_1 \mid (P \mid Q) \longrightarrow^*_{\mathbf{H}} R$ with $\overline{x} \in \mathit{SBarbs}(R)$. The required result follows, since $P \mid Q \preceq P_1 \mid (P \mid Q)$.
- Case for $P = (\nu y : \xi)\,P_1$: By the typing rules, we have $\Gamma, y : \xi/U \vdash_l P_1$. Since $rel((\Gamma, y : \xi/U) \mid \Delta$ and $\#(P_1) < \#(P)$ hold, we can apply induction hypothesis to obtain R' such that $P_1 \mid Q \longrightarrow^*_{\mathbf{H}} R'$ and $\overline{x} \in \mathit{SBarbs}(R')$. Thus, the required result holds for $R = (\nu y : \xi)\,R'$.
- Case for $P = \mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2$: By the assumption $rel(\Gamma)$, b is either $\mathit{true}^{l'}$ or $\mathit{false}^{l'}$. By the typing rules, $\Gamma \vdash_l P_i$ holds. So, by the induction hypothesis, there exists R such that $P_i \mid Q \longrightarrow^*_{\mathbf{H}} R$ and $\overline{x} \in \mathit{SBarbs}(R)$. The required result follows, since $P \preceq P_i$ for $i = 1$ or 2.

□

Theorem 5 *Let C be a finite-level context. If $\Gamma \vdash_l C[P]$ and $rel(\Gamma)$, then $C \longrightarrow^*_{\mathbf{H}} E$ for some evaluation context E.*

Proof The proof proceeds by induction on the depth of the hole in C. If the depth is 0, the result follows immediately (since C is itself an evaluation context). If C is of the form $E_1[\overline{x}\langle \widetilde{v} \rangle.\,C_1]$, then by Lemma 25, there exists R such that $E[\overline{x}\langle \widetilde{v} \rangle.\,C_1[P]] \longrightarrow^*_{\mathbf{H}} R$ with $x \in \mathit{Barbs}(R)$. So, $E_1[\overline{x}\langle \widetilde{v} \rangle.\,C_1] \longrightarrow^*_{\mathbf{H}} E'_1[C_1]$ for some evaluation context E'_1. By the induction hypothesis, there exists E such that $E'_1[C_1] \longrightarrow^*_{\mathbf{H}} E$. Therefore, we have $C \longrightarrow^*_{\mathbf{H}} E$ as required. The case where C is of the form $E_1[x(\widetilde{y}).\,C_1]$ is similar. □

B.4 Simulation of P by $\mathbf{Er}_\Gamma(P)$

In this section, we show that the behavior of P can be simulated by its erasure $\mathbf{Er}_\Gamma(P)$.

Lemma 26 *If $P \preceq Q$, then $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(Q)$.*

Proof This follows by straightforward induction on derivation of $P \preceq Q$. The only non-trivial is the case where S-IFT or S-IFF is applied.

- Case S-IFT: In this case, $P = \mathbf{if}\ \mathit{true}^{l'}\ \mathbf{then}\ Q\ \mathbf{else}\ R$. If $l' = \mathbf{L}$, then $\mathbf{Er}_\Gamma(P) = \mathbf{if}\ \mathit{true}^{l'}\ \mathbf{then}\ \mathbf{Er}_\Gamma(Q)\ \mathbf{else}\ \mathbf{Er}_\Gamma(R)$, so that we have $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(Q)$ as required. If $l' = \mathbf{H}$, then $\mathbf{Er}_\Gamma(P) = \mathbf{0}$. By the assumption $\Gamma \vdash_l \mathbf{if}\ \mathit{true}^{\mathbf{H}}\ \mathbf{then}\ Q\ \mathbf{else}\ R$, it must be the case that $\Gamma \vdash_{\mathbf{H}} Q$. So, by Lemma 4, we have $\mathbf{Er}_\Gamma(P) = \mathbf{0} \preceq \mathbf{Er}_\Gamma(Q)$ as required.
- Case S-IFF: Similar to the case for R-IFT.

□

Lemma 27 *Suppose* $\Gamma \vdash_l P$. *If* $P \longrightarrow_{\mathbf{H}} Q$, *then* $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(Q)$. *If* $P \longrightarrow_{\mathbf{L}} Q$, $\mathbf{Er}_\Gamma(P) \longrightarrow \mathbf{Er}_\Gamma(Q)$.

Proof This follows by induction on derivation for $P \longrightarrow_{l_1} Q$ with case analysis on the last rule used.

- Case R-COM: In this case, $P = \overline{x}\langle\widetilde{v}\rangle^{l_1,t_1}.\, P_1 \mid x(\widetilde{y})^{l_1,t_2}.\, P_2$ and $Q = P_1 \mid [\widetilde{y} \mapsto \widetilde{v}]P_2$. By the assumption $\Gamma \vdash_l P$, $\Gamma(x)$ is of the form $\langle\widetilde{\tau}\rangle^{l_1}/U$. If $l_1 = \mathbf{L}$, then $\mathbf{Er}_\Gamma(P) = \overline{x}\langle\widetilde{v}'\rangle.\, \mathbf{Er}_\Gamma(P_1) \mid x(\widetilde{y}).\, \mathbf{Er}_{\Gamma,\widetilde{y}:\widetilde{\tau}}(P_2)$ where $v_i' = \mathbf{ErV}_{\tau_i}(v_i)$. By Lemma 23, $\mathbf{Er}_\Gamma([\widetilde{y} \mapsto \widetilde{v}]P_2) = [\widetilde{y} \mapsto \widetilde{v}']\mathbf{Er}_{\Gamma,\widetilde{y}:\widetilde{\tau}}(P_2)$. So, we have $\mathbf{Er}_\Gamma(P) \longrightarrow \mathbf{Er}_\Gamma(Q)$ as required.
  If $l_1 = \mathbf{H}$, then $\mathbf{Er}_\Gamma(P) = \mathbf{Er}_\Gamma(P_1) \mid \mathbf{Er}_{\Gamma,\widetilde{y}:\widetilde{\tau}}(P_2)$. $\mathbf{Er}_\Gamma(Q) = \mathbf{Er}_\Gamma(P_1) \mid \mathbf{Er}_\Gamma([\widetilde{y} \mapsto \widetilde{v}]P_2)$. By the condition on well-formed types, $\mathbf{High}(\tau_i)$ holds. By Lemmas 23 and 24, we have:

  $$\begin{array}{l} \mathbf{Er}_\Gamma([\widetilde{y} \mapsto \widetilde{v}]P_2) \\ = [\widetilde{y} \mapsto \mathbf{ErV}_\Gamma(\widetilde{v})]\mathbf{Er}_{\Gamma,\widetilde{y}:\widetilde{\tau}}(P_2) \\ = \mathbf{Er}_{\Gamma,\widetilde{y}:\widetilde{\tau}}(P_2) \end{array}$$

  So, $\mathbf{Er}_\Gamma(P) = \mathbf{Er}_\Gamma(Q)$ holds.
- Case R-PAR: In this case, $P = P_1 \mid P_2$ and $Q = Q_1 \mid P_2$ with $P_1 \longrightarrow_{l_1} Q_1$. By the assumption $\Gamma \vdash_l P$, there must exist Γ_1 such that $\Gamma_1 \vdash_l P_1$ and $\Gamma \sim \Gamma_1$. By the induction hypothesis, $\mathbf{Er}_{\Gamma_1}(P_1) \longrightarrow \mathbf{Er}_{\Gamma_1}(Q_1)$ holds if $l_1 = \mathbf{L}$ and $\mathbf{Er}_{\Gamma_1}(P_1) \preceq \mathbf{Er}_{\Gamma_1}(Q_1)$ holds if $l_1 = \mathbf{H}$. So, $\mathbf{Er}_{\Gamma_1}(P) \longrightarrow \mathbf{Er}_{\Gamma_1}(Q)$ holds if $l_1 = \mathbf{L}$ and $\mathbf{Er}_{\Gamma_1}(P) \equiv_0 \mathbf{Er}_{\Gamma_1}(Q)$ holds if $l_1 = \mathbf{H}$. Since $\Gamma \sim \Gamma_1$, Lemma 21 implies $\mathbf{Er}_{\Gamma_1}(P) = \mathbf{Er}_\Gamma(P)$ and $\mathbf{Er}_{\Gamma_1}(Q) = \mathbf{Er}_\Gamma(Q)$. Thus, we have the required result.
- Case R-NEW: Trivial by the induction hypothesis.
- Case R-SPCONG: In this case, $P \preceq P'$, $P' \longrightarrow_{l_1} Q'$, and $Q' \preceq Q$. By Lemma 26, $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(P')$ and $\mathbf{Er}_\Gamma(Q') \preceq \mathbf{Er}_\Gamma(Q)$ hold. By Lemma 17, $\Gamma \vdash_l P'$. So, by the induction hypothesis, $\mathbf{Er}_\Gamma(P') \longrightarrow \mathbf{Er}_\Gamma(Q')$ holds if $l_1 = \mathbf{L}$, and $\mathbf{Er}_\Gamma(P') \equiv_0 \mathbf{Er}_\Gamma(Q')$ holds if $l_1 = \mathbf{H}$. Therefore, $\mathbf{Er}_\Gamma(P) \longrightarrow \mathbf{Er}_\Gamma(Q)$ holds if $l_1 = \mathbf{L}$ and $\mathbf{Er}_\Gamma(P) \equiv_0 \mathbf{Er}_\Gamma(Q)$ holds if $l_1 = \mathbf{H}$.

□

Lemma 28 *If* Γ *is a low-level type environment and* $\Gamma \vdash_l P$, *then* $\mathit{SBarbs}(P) \subseteq \mathit{SBarbs}(\mathbf{Er}_\Gamma(P))$.

Proof Suppose that $\overline{x} \in \mathit{SBarbs}(P)$. Then, $P \preceq (\nu\widetilde{y})\,(\overline{x}\langle\widetilde{v}\rangle.\, Q \mid R)$ with $x \notin \{\widetilde{y}\}$. By Lemma 26, $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma((\nu\widetilde{y})\,(\overline{x}\langle\widetilde{v}\rangle.\, Q \mid R))$ By the assumptions that Γ is a low-level type environment and that $\Gamma \vdash_l P$, $\Gamma(x)$ is of the form $\langle\widetilde{\tau}\rangle^{\mathbf{L}}/U$. So, $\mathbf{Er}_\Gamma((\nu\widetilde{y})\,(\overline{x}\langle\widetilde{v}\rangle.\, Q \mid R))$ is of the form $(\nu\widetilde{y}')\,(\overline{x}\langle\widetilde{v}'\rangle.\, Q' \mid R')$. Thus $\overline{x} \in \mathit{SBarbs}(\mathbf{Er}_\Gamma(P))$ holds as required. Similarly, $x \in \mathit{SBarbs}(P)$ implies $x \in \mathit{SBarbs}(\mathbf{Er}_\Gamma(P))$. □

B.5 Simulation of $\mathbf{Er}_\Gamma(P)$ by P

In this subsection, we show that the behavior of $\mathbf{Er}_\Gamma(P)$ can be simulated by the original process P.

Lemma 29 *Suppose that $rel(\Gamma)$ and $\Gamma \vdash_l Q$ hold. If $P_1 \preceq P_1'$ and $P_1 \mid P_2 \preceq \mathbf{Er}_\Gamma(Q)$, then there exists Q' such that $Q \longrightarrow^* \preceq Q'$ and $P_1' \mid P_2 \preceq \mathbf{Er}_\Gamma(Q')$.*

Proof The proof proceeds by induction on derivation of $P_1 \preceq P_1'$ with case analysis on the last rule used.

- Case for the rule for reflexivity: Since $P_1' = P_1$, the result follows for $Q' = Q$.
- Case for the rule for transitivity: In this case, $P_1 \preceq P_1'' \preceq P_1'$. By the induction hypothesis, there exists Q'' such that $Q \longrightarrow^* \preceq Q''$ and $P_1'' \mid P_2 \preceq \mathbf{Er}_\Gamma(Q'')$. By Theorem 1 and Lemmas 14 and 17, there exists Γ' such that $\Gamma' \vdash_l Q''$ and $rel(\Gamma')$ holds. By the induction hypothesis, there exists Q' such that $Q'' \longrightarrow^* \preceq Q'$ and $P_1' \mid P_2 \preceq \mathbf{Er}_{\Gamma'}(Q')$. By Lemma 21, we have $P_1' \mid P_2 \preceq \mathbf{Er}_\Gamma(Q')$ as required.
- Cases for S-ZERO1, S-ZERO2, S-ZERO3, S-COMMUT, S-ASSOC, S-NEW, and S-SWAP: Trivial. (Let $Q' = Q$.)
- Case for S-IFT: In this case, $P_1 = \mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ P_1'\ \mathbf{else}\ R$. If $P_1 \mid P_2 \preceq \mathbf{Er}_\Gamma(Q)$ has been derived from $P_1 \preceq P_1'$, then $Q' = Q$ satisfies the required condition. Otherwise, $\mathbf{Er}_\Gamma(Q) = E[P_1]$ and $[\,]_{FV(P_1)} \mid P_2 \preceq E$ for some evaluation context E. So, by the definition of $\mathbf{Er}$, $Q = C[\mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ Q_1\ \mathbf{else}\ Q_2]$ for some finite-level context C and Q_1 such that $\mathbf{Er}_\Gamma(C) = E$, $\mathbf{Er}_\Delta(Q_1) = P_1'$, and $\mathbf{Er}_\Delta(Q_2) = R$ where $\Delta = ext(\Gamma, C)$. By Theorem 5, $C \longrightarrow_{\mathbf{H}}^* E'$ for some evaluation context E'. Let $Q' = E'[Q_1]$. Then,

  $$\begin{array}{rcl} Q & = & C[\mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ Q_1\ \mathbf{else}\ Q_2] \\ & \longrightarrow_{\mathbf{H}}^* & E'[\mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ Q_1\ \mathbf{else}\ Q_2] \\ & \preceq & E'[Q_1] \\ & = & Q' \end{array}$$

  Moreover, by Lemma 27, $\mathbf{Er}_\Gamma(C[Q_1]) \preceq \mathbf{Er}_\Gamma(E'[Q_1]) = \mathbf{Er}_\Gamma(Q')$ holds, which implies $P_1' \mid P_2 \preceq E[P_1'] = \mathbf{Er}_\Gamma(C[Q_1]) \preceq \mathbf{Er}_\Gamma(Q')$.
- Case for S-IFF: Similar to the case for S-IFT.
- Case for S-REP: In this case, $P_1 = *P_{11}$ and $P_1' = *P_{11} \mid P_{11}$. If $P_1 \mid P_2 \preceq \mathbf{Er}_\Gamma(Q)$ has been derived from $*P_{11} \preceq *P_{11} \mid P_{11}$, then $Q' = Q$ satisfies the required condition. Otherwise, $\mathbf{Er}_\Gamma(Q) = E[P_1]$ and $[\,]_{FV(P_1)} \mid P_2 \preceq E$ for some evaluation context E. So, $Q = C[*Q_1]$ for some context context C and process Q_1, such that $\mathbf{Er}_\Gamma(C) = E$ and $\mathbf{Er}_\Delta(Q_1) = P_{11}$ for $\Delta = ext(\Gamma, C)$. If $P_{11} \equiv \mathbf{0}$, then $Q' = Q$ satisfies the required condition. Otherwise, by the definition of $\mathbf{Er}$, C is a finite-level context. By Theorem 5, $C \longrightarrow_{\mathbf{H}}^* E'$ holds for some evaluation context E'. Let $Q' = E'[*Q_1 \mid Q_1]$. Then, $Q = C[*Q_1] \longrightarrow^* \preceq E'[*Q_1] \preceq Q'$. Moreover, by Lemma 27, $\mathbf{Er}_\Gamma(C[*Q_1 \mid Q_1]) \preceq \mathbf{Er}_\Gamma(E'[*Q_1 \mid Q_1]) = \mathbf{Er}_\Gamma(Q')$ holds, which implies $P_1' \mid P_2 \preceq E[P_1'] = \mathbf{Er}_\Gamma(C[*Q_1 \mid Q_1]) \preceq \mathbf{Er}_\Gamma(Q')$.
- Case for S-PAR: In this case, $P_1 = P_{11} \mid P_{12}$ and $P_1' = P_{11}' \mid P_{12}$ with $P_{11} \preceq P_{11}'$. By the assumption $P_1 \mid P_2 \preceq \mathbf{Er}_\Gamma(Q)$, we have $P_{11} \mid (P_{12} \mid P_2) \preceq \mathbf{Er}_\Gamma(Q)$. By the induction hypothesis, there exists Q' such that $Q \longrightarrow^* \preceq Q'$ and $P_{11}' \mid (P_{12} \mid P_2) \preceq \mathbf{Er}_\Gamma(Q')$. The required result follows, since $P_1' \mid P_2 \preceq P_{11}' \mid (P_{12} \mid P_2) \preceq \mathbf{Er}_\Gamma(Q')$.
- Case for S-CNEW: In this case, $P_1 = (\nu x : \xi)\, P_{11}$ and $P_1' = (\nu x : \xi)\, P_{11}'$ with $P_{11} \preceq P_{11}'$. Let Q_1 be a process obtained by removing the prefix $(\nu x : \xi)$ from Q. Then, we have $P_{11} \mid P_2 \preceq \mathbf{Er}_{\Gamma, x : \xi/\mathbf{0}}(Q_1)$. By the induction hypothesis, there exists Q_1' such that $Q_1 \longrightarrow^* \preceq Q_1'$ and $P_{11}' \mid P_2 \preceq \mathbf{Er}_{\Gamma, x : \xi/\mathbf{0}}(Q_1')$. Let $Q' = (\nu x : \xi)\, Q_1$. Then, we have $Q \longrightarrow^* \preceq Q'$ and $P_1' \mid P_2 \preceq \mathbf{Er}_\Gamma(Q')$ as required.

□

Corollary 2 *Suppose $rel(\Gamma)$ and $\Gamma \vdash_l Q$. If $P \preceq \mathbf{Er}_\Gamma(Q)$ and $P \preceq P'$, then there exists Q' such that $Q \longrightarrow^* Q'$ and $P' \preceq \mathbf{Er}_\Gamma(Q')$.*

Proof Let $P_1 = P$, $P_1' = P'$, and $P_2 = \mathbf{0}$ in Lemma 29. Then, there exists Q' such that $Q \longrightarrow^* \preceq Q'$ and $P' \mid \mathbf{0} \preceq \mathbf{Er}_\Gamma(Q')$. From the second condition and S-ZERO1, $P' \preceq \mathbf{Er}_\Gamma(Q')$ follows. □

We write $\longrightarrow^+$ for the transitive closure of $\longrightarrow$.

Lemma 30 *Suppose $rel(\Gamma)$ and $\Gamma \vdash_l P$. If $P' \preceq \mathbf{Er}_\Gamma(P)$ and $P' \longrightarrow Q'$, then there exists Q such that $P \longrightarrow^+ Q$ and $Q' \preceq \mathbf{Er}_\Gamma(Q)$.*

Proof By Lemma 20, we have $P' \preceq (\nu\widetilde{u})\,(\overline{x}\langle\widetilde{v}\rangle.\, P_1' \mid x(\widetilde{y}).\, P_2' \mid P_3')$ and $(\nu\widetilde{u})\,(P_1' \mid [\widetilde{y} \mapsto \widetilde{v}]P_2' \mid P_3') \preceq Q'$ for some P_1', P_2', P_3'. By Corollary 2, there exists R such that $P \longrightarrow^* \preceq R$ and $(\nu\widetilde{u})\,(\overline{x}\langle\widetilde{v}'\rangle.\, P_1' \mid x(\widetilde{y}).\, P_2' \mid P_3') \preceq \mathbf{Er}_\Gamma(R)$. So, $\mathbf{Er}_\Gamma(R)$ is of the form $E[\overline{x}\langle\widetilde{v}'\rangle.\, P_1', x(\widetilde{y}).\, P_2']$ with $(\nu\widetilde{u})\,([]^{(1)}_{id,S_1} \mid []^{(2)}_{id,S_2} \mid P_3') \preceq E$ where $S_1 = FV(\overline{x}\langle\widetilde{v}'\rangle.\, P_1')$ and $S_2 = FV(x(\widetilde{y}).\, P_2')$. By the definition of $\mathbf{Er}$, $R = C[\overline{x}\langle\widetilde{v}\rangle.\, P_1, x(\widetilde{y}).\, P_2]$ for a finite-level context C with two holes such that $\mathbf{Er}_\Gamma(C) = E$ with $\mathbf{Er}_{\Delta_1}(\overline{x}\langle\widetilde{v}\rangle.\, P_1) = \overline{x}\langle\widetilde{v}'\rangle.\, P_1'$ and $\mathbf{Er}_{\Delta_2}(x(\widetilde{y}).\, P_2) = x(\widetilde{y}).\, P_2'$ where $\Delta_i = ext^{(i)}(\Gamma, C)$. By Lemma 5, there exists an evaluation context E' with two holes such that $C \longrightarrow^* E'$. Let $Q'' = E'[P_1, [\widetilde{y} \mapsto \widetilde{v}]P_2]$. Then $R \longrightarrow^+ Q''$ holds. Moreover, $(\nu\widetilde{u})\,(P_1' \mid [\widetilde{y} \mapsto \widetilde{v}']P_2' \mid P_3') \preceq E[P_1', [\widetilde{y} \mapsto \widetilde{v}']P_2'] = \mathbf{Er}_\Gamma(C[P_1, [\widetilde{y} \mapsto \widetilde{v}]P_2]) \preceq \mathbf{Er}_\Gamma(E'[P_1, [\widetilde{y} \mapsto \widetilde{v}]P_2]) = \mathbf{Er}_\Gamma(Q'')$. By Corollary 2 and $(\nu\widetilde{u})\,(P_1' \mid [\widetilde{y} \mapsto \widetilde{v}]P_2' \mid P_3') \preceq Q'$, there exists Q such that $Q' \preceq \mathbf{Er}_\Gamma(Q)$ and $Q'' \longrightarrow^* \preceq Q$. Moreover, we have $P \longrightarrow^* \preceq R \longrightarrow^+ Q'' \longrightarrow^* \preceq Q$, which implies $P \longrightarrow^+ Q$. □

Lemma 31 *If $rel(\Gamma)$ and $\Gamma \vdash_l P$, then SBarbs$(\mathbf{Er}_\Gamma(P)) \subseteq$ Barbs(P).*

Proof Suppose $\overline{x} \in$ *SBarbs*$(\mathbf{Er}_\Gamma(P))$. Then, $\mathbf{Er}_\Gamma(P) = E[\overline{x}\langle\widetilde{v}'\rangle P_1']$ for some evaluation context E. By the definition of $\mathbf{Er}$, $P = C[\overline{x}\langle\widetilde{v}\rangle P_1]$ for some finite-level context C and process P_1. By Theorem 5, $C \longrightarrow^* E'$ for some finite-level context. So, $P \longrightarrow^* E'[\overline{x}\langle\widetilde{v}\rangle P_1]$, which implies $\overline{x} \in$ *Barbs*(P). □

B.6 Proof of Theorem 4

Proof of Theorem 4 Let $\mathcal{R}$ be the set:

$\{(P, Q) \mid \Gamma \vdash_l P$ and $Q \preceq \mathbf{Er}_\Gamma(P)$ for some low-level, reliable $\Gamma\}$.

We show that $\mathcal{R}$ is a barbed bisimulation. Suppose $(P, Q) \in \mathcal{R}$, i.e., $\Gamma \vdash_l P$ and $Q \preceq \mathbf{Er}_\Gamma(P)$ for some low-level, reliable Γ. We check the three conditions of Definition 22.

- If $P \longrightarrow P'$, by Theorem 1, there exists Δ such that $\Gamma \longrightarrow^* \Delta$ and $\Delta \vdash_l P'$. Moreover, by Lemma 27, either $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(P')$ or $\mathbf{Er}_\Gamma(P) \longrightarrow \mathbf{Er}_\Gamma(P')$ holds. In the former case, let Q' be Q. Then, $Q' \preceq \mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(P') = \mathbf{Er}_\Delta(P')$. In the latter case, let Q' be $\mathbf{Er}_\Gamma(P')(= \mathbf{Er}_\Delta(P'))$. $Q \longrightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$ hold in either case.
- If $Q \longrightarrow Q'$, by Lemma 30, there exists P' such that $P \longrightarrow^+ P'$ and $Q' \preceq \mathbf{Er}_\Gamma(P')$. By Theorem 1, there exists Δ such that $\Gamma \longrightarrow^* \Delta$ and $\Delta \vdash_l P'$. By Lemma 21, $\mathbf{Er}_\Gamma(P') = \mathbf{Er}_\Delta(P')$. Therefore, we have $(P', Q') \in \mathcal{R}$ as required.
- Suppose that $\chi \in$ *Barbs*(P). Then there exists P' such that $P \longrightarrow^* P'$ and $\chi \in$ *SBarbs*(P'). By the first condition of barbed bisimulation, there exists Q' such that $Q \longrightarrow^* Q'$ and $Q' \preceq \mathbf{Er}_\Gamma(P')$. By Lemma 28, $\chi \in$ *Barbs*$(\mathbf{Er}_\Gamma(P')) =$ *SBarbs*(Q'). So, $\chi \in$ *Barbs*(Q) holds.
  On the other hand, suppose that $\chi \in$ *Barbs*(Q) holds. Then there exists Q' such that $Q \longrightarrow^* Q'$ and $\chi \in$ *SBarbs*(Q'). By the second condition of barbed bisimulation, there exists P' such that $P \longrightarrow^* P'$ and $Q' \preceq \mathbf{Er}_\Gamma(P')$. By Lemma 31, $\chi \in$ *Barbs*(P'). So, we have $\chi \in$ *Barbs*(P) as required. □